Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)

Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва інституту)

Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
Магістр
(освітньо-кваліфікаційний рівень)

на тему:

«ДОСЛІДЖЕННЯ МЕТОДІВ ТА РОЗРОБКА ЗАСОБІВ ЗАХИСТУ КОМП'ЮТЕРНИХ СИСТЕМ»

Виконав: студент групи 602-ТН
спеціальності 122 комп'ютерні науки
(шифр і назва напряму)
Кутенний С.І.
(прізвище та ініціали)

Керівник: к.т.н., доц. Головко Г.В.
(прізвище та ініціали)

Полтава – 2025 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВОГО ІНСТИТУТУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХГІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему:

«ДОСЛІДЖЕННЯ МЕТОДІВ ТА РОЗРОБКА ЗАСОБІВ ЗАХИСТУ КОМП'ЮТЕРНИХ СИСТЕМ

Студента групи 602-ТН Кутенного Сергія Ігоровича

Керівник роботи
кандидат технічних наук,
доцент Головко Г.В.

Завідувач кафедри
К.ф-м.н. доцент Двірна О.А.

Полтава – 2025 року

РЕФЕРАТ

Загальний обсяг роботи 109 с., 55 рисунки, 1 додаток, 1 таблицю, 27 бібліографічних найменувань.

Об’єкт дослідження: аналіз загроз та розробка антивірусного програмного забезпечення.

Предмет дослідження: антивірусне програмне забезпечення.

Мета роботи: розробка програмних засобів захисту комп’ютерних інформаційних систем.

Методи: аналіз вірусних загроз, розробка алгоритмів виявлення та нейтралізації шкідливого програмного забезпечення, тестування ефективності антивірусної програми.

Ключові слова: інформаційна безпека, захист інформації, антивірусна програма, вірусні загрози, виявлення та ліквідація вірусів.

ABSTRACT

Total volume of work 109 p., 55 figures, 1 appendice, 1 table, 27 bibliographic entries.

Object of research: threat analysis and development of antivirus software.

Subject of research: antivirus software.

Purpose of work: development of software tools for protecting computer information systems.

Methods: analysis of virus threats, development of algorithms for detecting and neutralizing malicious software, testing the effectiveness of the antivirus program.

Keywords: information security, information protection, antivirus program, virus threats, detection and elimination of viruses.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ ТА СКОРОЧЕНЬ

ІБ – інформаційна безпека.

КІС – комп’ютерна інформаційна система.

ЗІ – захист інформації.

ТЗІ – технічний захист інформації.

СЗІ – системи захисту інформації.

ПЗ – програмне забезпечення.

ОС – операційна система.

ЛОМ – локальна обчислювальна мережа.

МЗ – мережевий захист.

ШПЗ – шкідливе програмне забезпечення.

ДТ – державна таємниця.

ПС – програмна система.

RSA – алгоритм асиметричного шифрування.

AES – стандарт симетричного шифрування даних.

API – Інтерфейс програмування додатків.

IDE – Інтегроване середовище розробки.

SQL – Мова структурованих запитів для баз даних.

HTTPS – Протокол захищеної передачі гіпертексту.

URL — Uniform Resource Locator (уніфікований покажчик ресурсу).

Python – мова програмування високого рівня.

DoS/DDoS — Denial of Service / Distributed Denial of Service (відмова в обслуговуванні / розподілена відмова в обслуговуванні).

JSON — JavaScript Object Notation (формат обміну даними).

GUI — Graphical User Interface (графічний інтерфейс користувача).

ВСТУП

В умовах стрімкого розвитку цифрових технологій та зростання обсягів оброблюваної інформації захист комп'ютерних інформаційних систем стає одним із найважливіших завдань для сучасного суспільства. Конфіденційність, цілісність та доступність даних є основою ефективного функціонування як приватних підприємств, так і державних установ. Постійне ускладнення загроз у кіберпросторі вимагає вдосконалення існуючих методів захисту та розробки нових підходів до забезпечення інформаційної безпеки.

Сучасні комп'ютерні системи щоденно піддаються атакам: віруси, шкідливе програмне забезпечення, хакерські атаки та витоки інформації стають серйозним викликом для безпеки даних. Класичні методи захисту часто виявляються недостатньо ефективними проти нових видів загроз, що робить необхідним їхнє подальше вдосконалення. У цьому контексті особливо актуальним є дослідження ефективних методів та розробка нових засобів захисту комп'ютерних інформаційних систем.

У даній роботі досліджено сучасні методи захисту інформаційних систем. Особлива увага приділена використанню таких платформ, як VirusTotal, для аналізу загроз, а також розробці програмного продукту, який реалізує ці підходи в реальних умовах.

Значимість роботи полягає у створенні антивірусного програмного забезпечення, яке поєднує сучасні технології сканування та аналізу, забезпечуючи підвищений рівень захисту даних. Це не лише сприяє вирішенню актуальних задач у сфері кібербезпеки, але й створює фундамент для подальших досліджень і вдосконалення методів протидії загрозам.

Таким чином, кваліфікаційна робота магістра має як науково-теоретичну, так і прикладну цінність, спрямовану на покращення захисту комп'ютерних інформаційних систем у швидкозмінних умовах сучасного цифрового середовища.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМ ЗАХИСТУ КОМП'ЮТЕРНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Постановка задачі

Об'єкт дослідження: комп'ютерні інформаційні системи, зокрема їх захист від шкідливого програмного забезпечення.

Мета роботи: дослідження методів та розробка засобів захисту комп'ютерних систем від вірусних атак, створення антивірусного програмного додатка, який забезпечить захист від шкідливих програм і вірусів.

Методи дослідження: аналіз поточного стану захисту комп'ютерних систем, оцінка загроз і вразливостей, розробка програмного забезпечення для виявлення і нейтралізації шкідливих програм. У процесі розробки буде створено антивірусний додаток, що забезпечить ефективну перевірку системи на наявність шкідливих елементів і захист від них.

Основні задачі проекту:

1. Проведення комплексного аналізу існуючих загроз і вразливостей комп'ютерних систем:

- вивчення актуальних видів шкідливих програм і вірусів, які можуть загрожувати інформаційним системам;
- оцінка ефективності сучасних антивірусних рішень;
- визначення слабких місць у безпеці системи і виявлення потенційних точок проникнення шкідливих програм.

2. Розробка і тестування нового антивірусного додатка:

- створення алгоритмів для виявлення та нейтралізації шкідливих програм на основі традиційних сигнатурних методів;
- розробка інтерфейсу для зручності користувачів і автоматизації процесу сканування;

- забезпечення оновлення баз даних вірусних сигнатур для своєчасного виявлення нових загроз.

Таким чином, головною метою цього дипломного проекту є розробка інноваційного антивірусного додатка, який дозволить покращити захист комп'ютерних систем від вірусних атак і забезпечить надійний рівень безпеки даних.

1.2 Теоретичні основи інформаційної безпеки комп'ютерних систем (джерела загроз)

"Інформаційна безпека – це сукупність методів, технологій та заходів, які спрямовані на захист конфіденційних даних від несанкціонованого доступу, втручання, пошкодження або знищення", – [2]. Вона охоплює як фізичні методи безпеки, так і кібернетичні технології, такі як засоби контролю доступу, системи шифрування, рішення для захисту хмарних ресурсів (CASB) та технології інтеграції безпеки у процес створення програмного забезпечення (DevSecOps).

"Фундаментом інформаційної безпеки є три ключові принципи, відомі як **«тріада безпеки»:**

1. **Конфіденційність.**

"Конфіденційність полягає у забезпеченні доступу до інформації лише для авторизованих користувачів. Для реалізації цього принципу використовуються сучасні методи захисту, зокрема:

- шифрування даних;
- багатофакторна автентифікація;
- технології запобігання втраті даних (DLP)" – [5].

2. **Цілісність.**

"Принцип цілісності передбачає збереження незмінності та достовірності даних протягом їхнього життєвого циклу" – [5]. Щоб захистити дані від несанкціонованого внесення змін або пошкодження, застосовуються:

- контроль доступу до інформації;
- системи управління ідентифікацією користувачів;
- механізми журналювання та моніторингу змін у даних.

3. Доступність.

“Доступність забезпечує можливість своєчасного доступу до інформації для авторизованих користувачів. Для цього важливо підтримувати працездатність обладнання, регулярне оновлення систем та резервне копіювання даних” – [5].

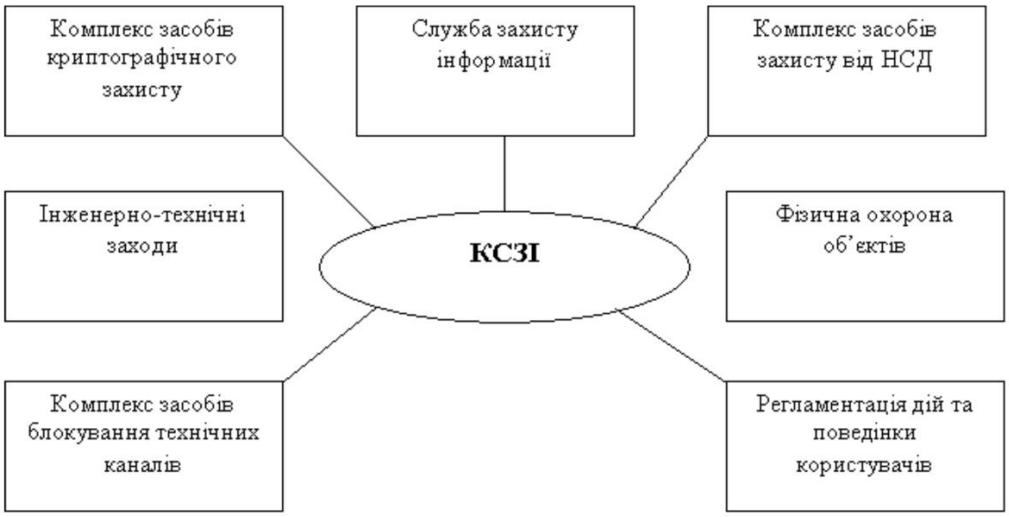

Рисунок 1.1 - Комплексна структура захисту інформації

“З розвитком інформаційного суспільства комп’ютерні інформаційні системи стали ключовим елементом наукової, комерційної та технологічної діяльності. Такі системи об’єднують апаратні, програмні та мережеві компоненти для обробки, зберігання і передачі даних” – [4].

“Апаратна частина охоплює процесори, пристрої зберігання даних, мережеві адаптери та інші фізичні компоненти” – [4].

“Програмне забезпечення включає операційні системи, прикладні програми та бази даних, що відповідають за управління інформаційними потоками” – [4].

Сьогодні значна увага приділяється загрозам для інформаційних систем, які походять із зовнішніх та внутрішніх джерел. Вони включають:

- шкідливе програмне забезпечення (віруси, трояни, шпигунські програми);
- несанкціонований доступ з боку зловмисників;
- вразливості програмного коду;
- фізичні пошкодження обладнання або програмного забезпечення;
- збої в мережевих інфраструктурах.

“Комплексний підхід до захисту інформаційних систем базується на ретельному аналізі загроз, оцінці ризиків і впровадженні інноваційних рішень, зокрема програмних продуктів для боротьби зі шкідливими програмами” - [7].

Таким чином, розвиток інформаційної безпеки в сучасних умовах вимагає системного підходу, який враховує як технічні, так і організаційні аспекти захисту. Реалізація цих заходів є основою для забезпечення стійкості та надійності комп’ютерних систем у контексті глобальних загроз.

1.3 Правові основи захисту інформації в Україні

В Україні захист інформації регулюється на державному рівні через закони, постанови та нормативно-технічні документи:

- Закон України «Про інформацію» від 02.10.1992 №2657-XII;
- Закон України «Про захист інформації в інформаційно-телекомунікаційних системах» від 05.07.1994 №80/94-ВР;
- Закон України «Про державну таємницю» від 21.01.1994 №3855-XII;
- Закон України «Про захист персональних даних» від 01.06.2010 №2297-VI;
- Постанова КМУ «Про затвердження Правил забезпечення захисту інформації...» від 29.03.2006 №373;
- Постанова КМУ «Про затвердження Інструкції обліку і використання документів із службовою інформацією» від 27.11.1998 №1893.

Також діють стандарти технічного захисту інформації (НД ТЗІ та ДСТУ), серед яких:

- НД ТЗІ 3.7-003-05,
- ДСТУ 3396.1-96,
- НД ТЗІ 2.5-008-02 та інші нормативи у сфері технічного контролю інформації.

1.4 Потреба в систематичному захисті інформації

Забезпечення захисту інформації є одним з найважливіших завдань сучасних підприємств та організацій. Компаніям необхідно не лише запобігати витокам даних чи втратам інформації, а й ефективно протидіяти атакам зловмисників та вірусному програмному забезпеченню.

“Проте одне універсальне рішення для всіх випадків не існує. Необхідно постійно адаптувати методи захисту до нових викликів, зокрема:

- впроваджувати інноваційні технології;
- відстежувати зростаючі можливості хакерів і зловмисників;
- регулярно оновлювати системи безпеки” – [3].

“Для великих підприємств важливим є не тільки захист електронних носіїв даних, а й впровадження **фізичних методів безпеки**, а саме:

- обмеження доступу до інформації на основі посадових обов’язків;
- зонування території з рівнями безпеки;
- забезпечення контролю доступу між підрозділами” – [4].

1.5 Класифікація загроз комп’ютерних інформаційних систем

“Загрози комп’ютерних інформаційних систем можна класифікувати за різними критеріями. Одним із основних є їхня **природа виникнення,** яка поділяється на:

- природні загрози – виникають через об’єктивні фізичні явища та стихійні лиха, що не залежать від людської діяльності;
- штучні загрози – є результатом дій людини, зокрема в сфері інформаційної безпеки” – [2].

“Залежно від рівня умислу, загрози поділяють на” - [2]:

- **“випадкові (ненавмисні)** – виникають через помилки чи недбалість користувачів або обслуговуючого персоналу. Вони можуть спричинити суттєві втрати інформації, зокрема до 80% усіх збитків. Приклади таких загроз включають:

– технічні збої (відключення електроенергії, неполадки серверів чи мережевих пристроїв);

– помилки персоналу при обробці чи збереженні даних;

– неправильна робота програмного забезпечення;

– видалення, зміна або втрата інформації через збої чи людський фактор;

– поширення вірусів;

– несанкціонований доступ;

– витік конфіденційної інформації” – [2].

- **“навмисні загрози** – є результатом цілеспрямованих дій зловмисників. Вони охоплюють:

– методи шпигунства та диверсій – крадіжка документів, підкуп, шантаж, аналіз відходів носіїв інформації;

– несанкціонований доступ через відсутність систем розмежування доступу;

– використання електромагнітних випромінювань для отримання інформації;

– модифікація даних або систем шляхом несанкціонованого впливу;

– атаки шкідливого програмного забезпечення” – [2].

“Непрямі загрози можна поділити на:

- природні – стихійні лиха, магнітні бурі, радіаційне випромінювання;

- людські фактори – зловмисне введення агентів серед персоналу, підкуп, шантаж, видалення або копіювання секретних даних, компрометація паролів чи ключів шифрування” - [2].

“Загрози через програмно-апаратні засоби включають:

- використання нелегального або неліцензійного програмного забезпечення, що може призвести до витоку ресурсів системи;
- зараження систем вірусами з руйнівними можливостями” - [2].

“Загрози за місцем походження:

- зовнішні загрози – джерела, розташовані поза межами захищеної зони. Приклади: перехоплення сигналів, аналіз даних, дистанційне спостереження;
- внутрішні загрози – виникають у межах контрольованої території, зокрема крадіжка документів чи пошкодження інфраструктури (живлення, охолодження, комунікації);
- загрози через периферійні пристрої – джерела, що мають доступ до периферійних компонентів;
- загрози зсередини системи – загрози, що виникають через архітектуру системи, внутрішні помилки чи неправильну експлуатацію” - [2].

“За взаємозв’язком із активністю системи:

1. Пасивні загрози – не змінюють структуру системи, наприклад, копіювання даних.

2. Активні загрози – модифікують структуру або зміст системи. Приклади:

- впровадження програмних закладок, вірусів;
- дії, що порушують функціонування системи;
- несанкціоновані зміни даних” - [2].

“За способом доступу до системи:

1. Стандартний шлях доступу:

- крадіжка паролів через перехоплення чи підбір;

- маскарад – використання імені іншого користувача для отримання повноважень.

2. Нестандартний шлях доступу:

- обхід заходів захисту через стороннє ПЗ чи недокументовані функції ОС” - [2].

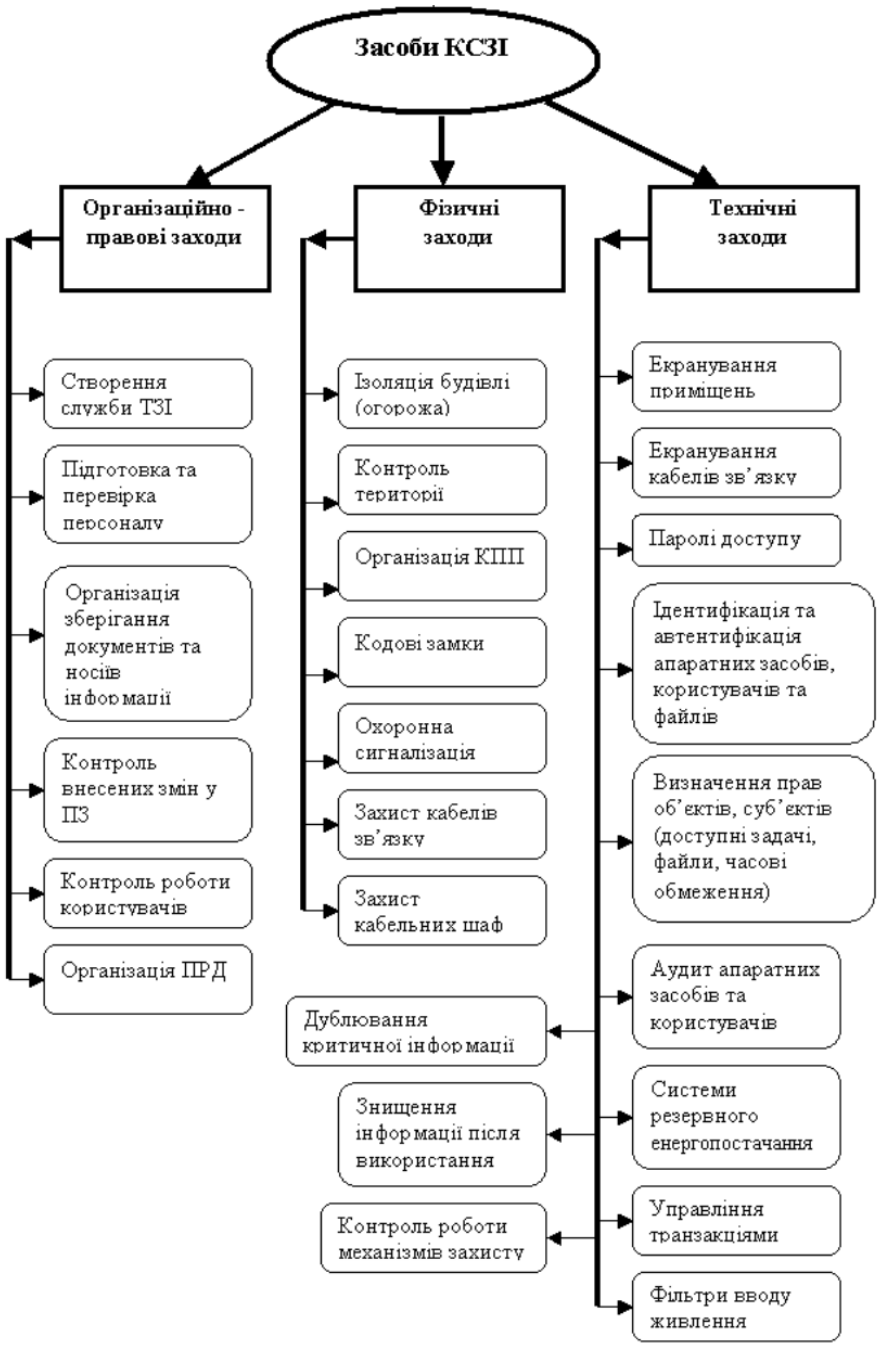

Рисунок 1.2 – Засоби комплексної системи захисту інформації

1.6 Класифікація атак на автоматизовані системи

1. Віддалений контроль комп’ютера через мережу (NetBus, BackOrifice).
2. Локальне проникнення (наприклад, GetAdmin).
3. Віддалена відмова в обслуговуванні (Teardrop, trin00).
4. Локальна відмова в обслуговуванні – перенавантаження ресурсів системи.
5. Сканування мережі для збору даних про структуру мережі (nmap).
6. Виявлення вразливостей (SATAN, Nessus, Xspider).
7. Злам паролів (L0phtCrack, Crack).
8. Пасивне прослуховування мережі – отримання конфіденційних даних (tcpdump, LanExplorer).

1.7 Сучасний стан систем інформаційного захисту

“Локальна мережа є специфічним прикладом розподіленої системи, головними характеристиками якої є кількість підключених робочих місць та їхнє територіальне розташування. Розподілені системи, як правило, складаються з численних комп'ютерів, сегменти яких можуть бути географічно віддаленими. Основною вразливістю є канали передачі даних, оскільки інформація під час транспортування по загальних лініях зв'язку може бути перехоплена, що загрожує порушенням таких ключових властивостей даних, як **конфіденційність** (доступ третіх осіб), **цілісність** (несанкціоновані зміни) і **доступність** (затримки або втрата даних)” - [18].

Крім цього, загрози можуть виникати внаслідок інформаційних атак через глобальну мережу. Такі атаки реалізуються за різними сценаріями, серед яких варто виділити:

- вірусні атаки;

- мережеві атаки, що мають на меті проникнення у систему або виведення її з ладу.

“Для ефективного захисту необхідно впроваджувати програмні, апаратні та комбіновані системи. Зокрема, **VPN (віртуальна приватна мережа)** забезпечує безпеку каналів зв'язку завдяки шифруванню даних“ - [18]. “Захист від мережевих загроз забезпечується такими рішеннями, як системи виявлення вторгнень і міжмережеві екрани. Окрему роль відіграють антивірусні програми, що борються зі шкідливим програмним забезпеченням (ПЗ)” - [19].

“Системи виявлення вторгнень функціонують на основі аналізу вхідного та вихідного трафіку, застосовуючи технології сигнатурного аналізу та виявлення аномалій. Серед популярних рішень варто згадати «Cisco IPS Sensor» серії 4500 і «McAfee Network Security Platform»” - [8].

“Антивірусні програми запобігають та ліквідують наслідки зараження шкідливим ПЗ. Основний виклик полягає у постійній модифікації вірусів, що дозволяє їм уникати детектування. Для підвищення ефективності використовуються методи статистичного та евристичного аналізу, а також регулярне оновлення баз даних вірусів. До найбільш популярних антивірусних рішень належать: Norton 360, NOD32, Avast, Dr. Web Security, Avira і 360 Security” - [8].

“**Сучасні методи захисту інформації базуються на таких принципах**:

1. Системний підхід — поєднання програмних, апаратних, фізичних і організаційних методів захисту на всіх етапах роботи з інформацією.
2. Постійне вдосконалення — адаптація системи до нових загроз і каналів витоку інформації.
3. Забезпечення надійності — контроль функціональності та стійкості системи до збоїв і атак.
4. Контроль працездатності — вдосконалення методів моніторингу захисних механізмів.

5. Боротьба з шкідливим ПЗ — ефективне виявлення та нейтралізація вірусів.

6. Оптимізація витрат — економічно обґрунтована організація захисних заходів” - [19].

Рисунок 1.3 – Комплексна система захисту

1.8 Джерела загроз та способи реалізації

“Забезпечення безпеки комп'ютерних систем під час передачі даних є ключовим завданням сучасного цифрового середовища. Загрози інформаційній безпеці можна класифікувати за кількома категоріями:

1. Перехоплення даних (Eavesdropping).

“Перехоплення інформації під час її передачі через мережу. Зловмисники можуть використовувати:

- аналіз пакетів даних;
- спеціалізовані програми для збору конфіденційної інформації.

Рисунок 1.4 - Перехоплення даних (Eavesdropping)

Захист: використання шифрування даних та захищених каналів передачі” – [9].

2. Атака "Man-in-the-Middle" (MITM).

“Зловмисник «вклинюється» між двома сторонами обміну даними, отримуючи можливість переглядати, модифікувати або блокувати передачу інформації.

Рисунок 1.5 - Атака "Man-in-the-Middle"

Захист: застосування криптографічних протоколів (TLS/SSL) та автентифікації" – [9].

3. Вторгнення у систему (Intrusion).

"Використання вразливостей у системі для отримання несанкціонованого доступу до даних.

Рисунок 1.6 - Вторгнення у систему (Intrusion)

Захист: регулярне оновлення програмного забезпечення та використання систем виявлення вторгнень" – [9].

4. Віруси та шкідливе ПЗ.

"Шкідливі програми, що призводять до втрати контролю над даними та витоку інформації.

Захист: антивірусні програми, евристичний аналіз та регулярні оновлення сигнатур" – [9].

5. Неавторизований доступ.

"Загроза, що виникає через недостатньо надійні механізми автентифікації та авторизації.

Захист: впровадження багатофакторної автентифікації" – [9].

6. Соціальна інженерія.

"Атаки, що використовують маніпуляції та психологічний вплив на людей для отримання конфіденційної інформації (фішинг, обман).

Захист: навчання персоналу та підвищення обізнаності у сфері кібербезпеки" – [9].

7. Атаки на відмову в обслуговуванні (DoS/DDoS).

"Перевантаження системи великою кількістю запитів для блокування її нормальної роботи.

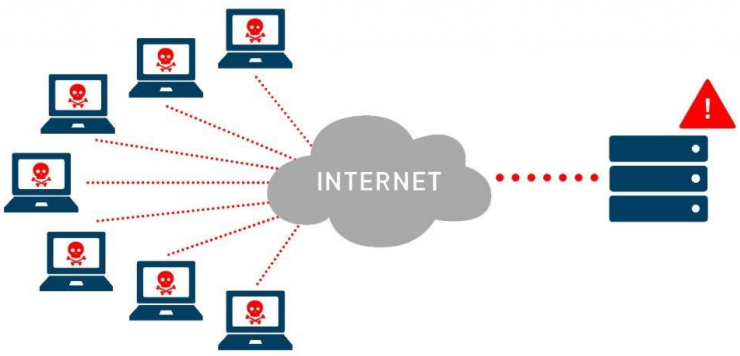

Рисунок 1.7 - Атаки на відмову в обслуговуванні (DoS/DDoS).

Захист: використання систем фільтрації трафіку та захисних шлюзів" – [9].

8. Недостатньо захищені канали передачі даних.

"Проблеми конфігурації мережевих пристроїв можуть призвести до витоку інформації.

Захист: правильне налаштування мережевого обладнання та шифрування даних" – [9].

9. USB-атаки.

“Несанкціоновані пристрої USB можуть використовуватись для введення шкідливого ПЗ чи крадіжки даних.

Захист: блокування USB-портів та контроль доступу до зовнішніх носіїв” – [9].

10. Використання хмарових сервісів без дозволу.

“Зберігання даних у незахищених хмарових сховищах може призвести до витоку конфіденційної інформації.

Захист: політики безпеки для хмарових технологій та моніторинг використання ресурсів” – [9].

11. “Мережева розвідка та сніфінг.

Зловмисники використовують:

- сканування портів**;**
- DNS-запити**;**
- режим **promiscuous mode**, щоб перехопити пакети даних.

Рисунок 1.8 - Мережева розвідка та сніфінг

Захист: обмеження доступу до критичних портів та моніторинг мережевої активності ” – [10].

12. IP-підробка (Spoofing).

“Використання підроблених IP-адрес для імітації дій авторизованого користувача.

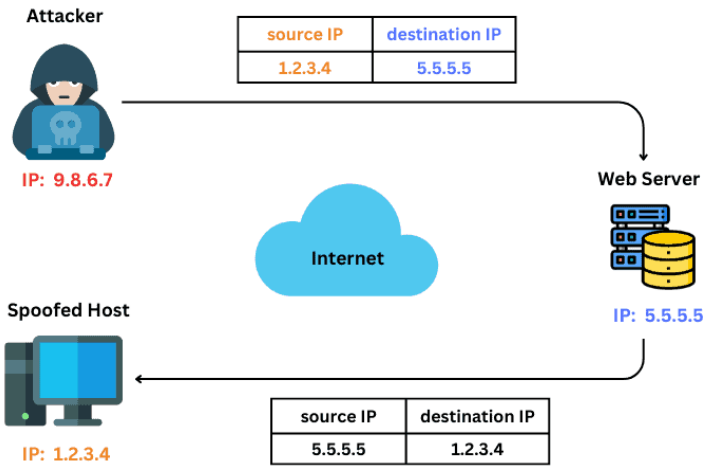

Рисунок 1.9 - IP-підробка (Spoofing)

Захист: перевірка джерел запитів та впровадження додаткових рівнів автентифікації" – [9].

13. Ін'єкції.

Використання шкідливих команд для впливу на систему. Різновиди:

- SQL-ін'єкція — атака на бази даних;
- PHP-ін'єкція — атака на веб-додатки;
- міжсайтовий скриптинг (XSS) — впровадження коду у веб-сторінки;
- XPath-ін'єкція — маніпуляція запитами до XML-баз даних.

Рисунок 1.10 – Ін'єкції

Захист: застосування валідації даних та використання веб-фільтрів.

"Для забезпечення захисту інформаційних систем від зазначених загроз слід:

- використовувати **шифрування даних**;

- впроваджувати **багатофакторну автентифікацію**;
- регулярно **оновлювати ПЗ** та **апаратне забезпечення**;
- підвищувати **обізнаність персоналу** у питаннях безпеки;
- розробляти та вдосконалювати **політики інформаційної безпеки**” – [9].

Дотримання цих заходів дозволить мінімізувати ризики, пов'язані з кіберзагрозами.

РОЗДІЛ 2
АНАЛІЗ АНТИВІРУСНИХ ПРОГРАМ

2.1 Антивірусні програми

“**Антивірусні програми** – це спеціалізовані засоби, що призначені для виявлення, видалення комп'ютерних вірусів і шкідливого ПЗ, а також відновлення заражених файлів. Вони також виконують профілактичну функцію, запобігаючи модифікації файлів, системи шкідливим кодом” - [9].

“Загальні методи захисту від «електронних інфекцій» включають резервне копіювання даних, обмеження доступу до інформації та профілактичні заходи. Крім того, набуває популярності використання апаратних рішень, зокрема спеціалізованих антивірусних карт, які інтегруються у слоти розширення комп’ютера. Однак основним інструментом захисту залишається антивірусне програмне забезпечення, яке можна класифікувати на кілька категорій” – [10]:

1. **Програми-детектори.**

“Ці програми виявляють віруси шляхом пошуку їх сигнатур у базі даних. Однак вони обмежені лише вже відомими вірусами й потребують регулярного оновлення. Хоча деякі детектори можуть виявляти нові типи вірусів, вони не здатні заздалегідь розпізнати невідомі загрози. Багато детекторів також мають функцію лікування заражених файлів” – [10].

2. **Програми-доктори (фаги)**.

“Це антивірусні засоби, що не лише виявляють віруси, а й лікують файли, відновлюючи їх початковий стан після видалення шкідливого коду. Окремо виділяють **поліфаги** – програми-доктори, які можуть обробляти широкий спектр вірусів. Як і детектори, вони потребують регулярного оновлення для ефективної роботи” – [10].

3. **Програми-ревізори.**

“Ці програми фіксують стан системи (файлів, каталогів тощо) до зараження вірусом і порівнюють його з поточним станом після завантаження ОС. Вони можуть виявляти зміни, включно зі стелс-вірусами, і, у деяких випадках, відновлювати початкові параметри” – [10].

4. Програми-фільтри (сторожі).

“Це резидентні антивірусні програми, що контролюють дії на комп’ютері й попереджають користувача про потенційно шкідливі операції. Вони дозволяють запобігти вірусам ще на ранній стадії, хоча іноді можуть конфліктувати з іншими програмами” – [10].

5. Програми-вакцини (імунізатори).

“Вакцини змінюють структуру програм або дисків так, щоб вони імітували заражені файли. Цей метод ефективний лише проти вже відомих вірусів” – [10].

“Основні заходи антивірусного захисту

Для забезпечення безпеки комп’ютерної системи слід дотримуватись наступних рекомендацій:

1. Використання комплексного антивірусного ПЗ.

Регулярне оновлення антивірусних програм забезпечує максимальну ефективність у боротьбі з новими загрозами.

2. Перевірка системи на наявність вірусів.

Необхідно регулярно сканувати файли, системні області та оперативну пам'ять. Для глибокої перевірки рекомендується завантажувати ОС із захищених носіїв.

3. Перевірка зовнішніх носіїв і файлів.

Перед використанням дискет або файлів, отриманих з ненадійних джерел, слід перевіряти їх на наявність вірусів.

4. Захист носіїв від запису.

Забороніть запис на дискети, що не потребують модифікації, щоб уникнути зараження важливих даних.

5. Архівування даних.

Регулярно створюйте резервні копії критично важливої інформації та обмежуйте доступ до них для сторонніх осіб” – [12].

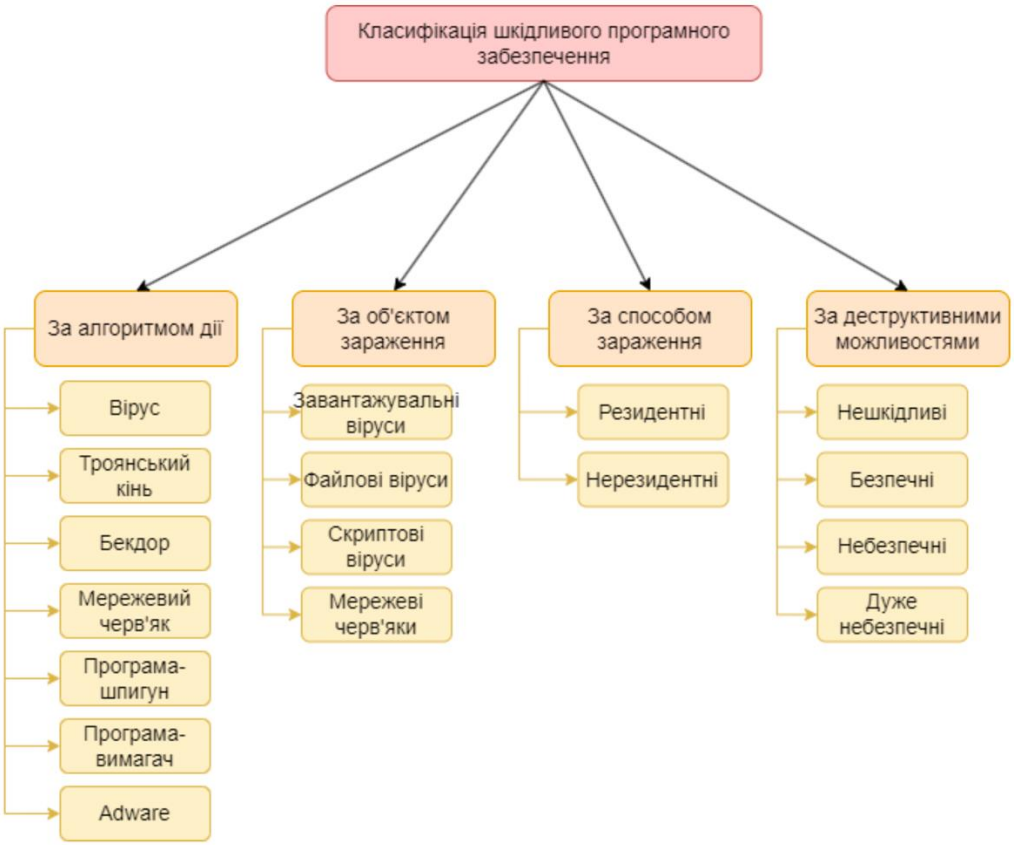

Рисунок 2.1 – Класифікація шкідливого ПЗ

2.1.1 Основні способи антивірусного захисту:

1. Шифрувальне програмне забезпечення.

“Шифрувальні програми забезпечують захист конфіденційних даних шляхом їх перетворення у зашифрований формат із використанням спеціальних алгоритмів. Доступ до такої інформації можливий лише при наявності відповідного ключа” - [11].

2. Мережевий екран (фаєрвол).

“Фаєрволи контролюють і фільтрують мережевий трафік відповідно до встановлених правил безпеки” – [11]. Основні типи:

- мережевий рівень – забезпечує базову фільтрацію на мережевому та транспортному рівнях;

- прикладний рівень – обробляє дані на рівні застосунків, забезпечуючи більш точний контроль;
- рівень з’єднання – обслуговує різноманітні протоколи, надаючи ширші можливості захисту.

3. Система виявлення вторгнень (IDS).

“IDS виявляє несанкціонований доступ і загрози у системах або мережах” – [11]. Вони поділяються на:

- мережеві IDS (NIDS) **–** аналізують мережевий трафік;
- хостові IDS (HIDS) – відстежують активність на конкретних пристроях.

IDS може використовувати сигнатурний аналіз (пошук шаблонів) та аналіз аномалій (виявлення відхилень у поведінці системи).

4. Пісочниця (Sandbox).

“Пісочниця – це ізольоване середовище для безпечного виконання підозрілого коду. Вона використовується для захисту системи та у процесі розробки програмного забезпечення” – [11].

5. Система управління інформаційною безпекою (СУІБ).

“СУІБ – це комплексний підхід до управління безпекою, що включає планування, впровадження, моніторинг та вдосконалення заходів інформаційного захисту. Вона ґрунтується на циклі «плануй-виконуй-перевіряй-дій»” – [11].

6. SIEM-системи.

“SIEM забезпечують збір, аналіз та зберігання даних для виявлення загроз у реальному часі” – [11]. Основні функції включають:

- агрегацію даних з різних джерел;
- кореляцію подій для виявлення загроз;
- сповіщення про проблеми безпеки;
- зберігання даних для подальшого аналізу.

“Застосування комплексного підходу до антивірусного захисту та систем управління безпекою дозволяє знизити ризики й забезпечити стабільну роботу комп'ютерних систем” – [11].

2.1.2 Комп’ютерні віруси та їх різновиди. “Комп’ютерні віруси є однією з найпоширеніших загроз у світі інформаційних технологій. Це спеціально створені шкідливі програми, основна мета яких – уразити виконувані файли, документи, фото, дані та інші ресурси системи. Внаслідок їх діяльності виникає серйозна втрата даних або зниження продуктивності програм і всієї операційної системи” – [10].

Рисунок 2.2 – Класифікація вірусів

1. Файлові віруси.

“Файлові віруси вражають окремі виконувані файли, перезаписуючи частину коду або вставляючи заражені фрагменти в структуру оригінального файлу. Ці віруси можуть розмножуватись на різних операційних системах і передаватися між пристроями. Часто такі віруси називають *паразитичними*, оскільки система сприймає їх як невід'ємну частину програми. В результаті

вірус отримує ті ж права, що й легітимна програма, що дозволяє йому створювати власні копії, залишатися в пам’яті та змінювати функціонування ОС” – [10].

“Фази роботи файлового вірусу:

- фаза сну (бездіяльності): вірус знаходиться у прихованому стані, не проявляючи активності, щоб уникнути виявлення антивірусами;
- фаза розмноження: при першому запуску зараженого файлу активується вірусний код, що запускає процес копіювання вірусу;
- тригерна фаза: вірус переходить до виконання активних дій;
- фаза виконання: вірус виконує шкідливі операції – видаляє дані, завдає шкоду файлам чи системі, виводить небажані повідомлення” – [10].

2. Макровіруси.

“Макровіруси – це специфічні віруси, що пишуться мовою макросів (наприклад, VBA у Microsoft Word та Excel). Вони функціонують не в середовищі операційної системи, а в додатках, які підтримують макроси. Макровіруси зазвичай поширюються через текстові та табличні документи, а також файли, надіслані електронною поштою або отримані з небезпечних вебресурсів” – [10].

“Особливості роботи макровірусу:

- поширюється лише у програмах, які підтримують макромови;
- використовує шаблони макросів для автоматичного перенесення заражених кодів між файлами;
- при відкритті документа вірус активується та створює власні макроси, які зберігаються у глобальних шаблонах (наприклад, у DOT-файлах Word). Це дозволяє вірусу інфікувати інші документи під час їх відкриття або збереження” – [10].

3. Завантажувальні віруси.

“Завантажувальні віруси заражають сектори завантаження носіїв (гнучкі диски, жорсткі диски) та активуються під час старту операційної системи.

Вірусний код копіюється з інфікованого носія в пам’ять комп’ютера, поширюючись на інші диски” – [12].

“Особливістю цих вірусів є складність видалення, оскільки багато антивірусних програм не можуть очистити завантажувальні сектори (MBR) під час роботи ОС. Для лікування часто потрібні спеціалізовані завантажувальні антивірусні диски.

Наслідки інфекції:

- збої у роботі системи;
- неможливість коректного завантаження ОС (наприклад, помилка «Недійсний системний диск»);
- пошкодження або втрата даних” – [12].

4. Поліморфні віруси.

“Поліморфні віруси – це складні віруси, які під час кожного зараження змінюють свій код за допомогою шифрування. Вони уникають виявлення антивірусами, які використовують сигнатурні методи аналізу” - [12].

Механізм роботи поліморфного вірусу:

- використання різних методів шифрування та випадкових ключів для маскування;
- мутація дешифруючого коду при кожному новому зараженні;
- поширення через електронні листи, небезпечні вебсторінки та інші ненадійні ресурси.

5. Троянські програми (Трояни).

“Троянський кінь, на відміну від вірусів і черв’яків, не поширюється самостійно. Він маскується під легітимну програму, щоб обдурити користувача та запустити шкідливий код. Після активації троян може змінювати файли, передавати конфіденційну інформацію або надавати зловмисникам доступ до системи.

Типи троянів:

- троян-завантажувач – завантажує інші шкідливі програми на заражений комп'ютер;

- троян задньої двері (Backdoor): створює прихований канал для віддаленого управління ПК;
- банківський троян: викрадає фінансову інформацію (банківські реквізити, паролі);
- ігровий троян: орієнтований на крадіжку облікових даних геймерів" - [12].

6. Віруси-шифрувальники.

"Віруси-шифрувальники блокують доступ до даних шляхом шифрування файлів та вимагають викуп за їх розшифрування. Після зараження вірус активується при перезавантаженні ОС, шифрує важливі файли та виводить повідомлення про викуп" - [12].

Спочатку ці віруси атакували тільки ОС Windows, але сьогодні вони можуть вражати й Linux, MacOS та Android.

7. Комп'ютерні черв'як.

"Черв'як – це саморозповсюджувальні програми, які атакують комп'ютери через мережі, використовуючи вразливості в системах. Вони створюють копії себе і поширюються без втручання користувача.

Способи поширення черв'яків:

- посилання на заражених вебсторінках або в електронних листах;
- вкладення в електронних листах;
- мережеві пакети, які проникають у пам'ять пристрою і активують код.

Черв'яки часто використовують методи шифрування для маскування своєї діяльності, завдаючи значної шкоди системам" - [12].

2.2 Алгоритми розпізнавання вірусів

"Антивірус розпочинає свою роботу, як тільки користувач увімкне операційну систему. Через значне накопичення файлів і активну роботу програмного забезпечення антивірус виконує трудомістку задачу порівняння

послідовностей коду. Це призводить до уповільнення роботи персонального комп’ютера. Саме тому зараз активно розробляють методи створення антивірусів із мінімальним впливом на системні ресурси” - [10].

1. “Метод "Signature Detection" (пошук за підписами).

Цей метод передбачає ретельну перевірку файлів, які додаються до системи або вже існують у ній, на наявність вірусів. Антивірусні програми мають попередньо встановлені бази даних вірусних підписів, які зберігають шаблони та унікальні біти коду вірусів. Під час роботи програма порівнює ці підписи з кодами файлів і вебресурсів. Якщо відбувається збіг, файл ізолюється у карантин для запобігання поширенню загрози” - [10].

2. “Евристичне виявлення.

Евристичний алгоритм є методом, що дозволяє знаходити прийнятні рішення серед багатьох альтернатив. Хоча такі рішення не завжди є оптимальними, вони забезпечують швидкість і ефективність.

Основні характеристики евристичних алгоритмів:

- швидкий пошук наближених рішень;
- можливість виявлення нових, невідомих вірусів;
- можливі хибні позитивні спрацьовування (заражені файли вважаються "здоровими" і навпаки)” - [10].

3. “Метод поведінкового виявлення.

Цей метод базується на спостереженні за поведінкою програм у реальному часі. Антивірус сканує активні програми та попереджає користувача про підозрілу активність, наприклад:

- заміну налаштувань файлів, не пов’язаних із додатком;
- спроби дистанційного підключення невідомими засобами;
- масове видалення або зміна файлів без очевидної причини.

Основні переваги методу:

- аналіз поведінки програм, а не лише їхнього коду;
- виявлення аномалій у реальному часі;

- використання машинного навчання для адаптації до нових загроз” - [10].

4. “Виявлення вірусів у пісочниці (sandboxing).

Метод полягає в ізоляції та аналізі виконання програм у **віртуальному середовищі** перед наданням їм доступу до основної системи.

Основні етапи методу включають:

- ізоляцію програми у віртуальному середовищі;
- моніторинг поведінки програми на предмет аномальних дій;
- аналіз спроб змінити конфігурацію системи або структуру файлів.

Цей метод забезпечує високий рівень безпеки, проте є трудомістким і повільним. Саме тому він активно використовується у корпоративних рішеннях для забезпечення цілісності систем” – [10].

5. “Видобуток даних (data mining).

На основі аналізу коду файлів та алгоритмів інтелектуального аналізу даних, антивірус визначає потенційну загрозу. Використання методів машинного навчання дозволяє підвищити ефективність виявлення шкідливого програмного забезпечення” - [10].

2.3 Порівняльний аналіз існуючих антивірусних програм

Порівняльний аналіз антивірусних програм передбачає детальне вивчення функціональних можливостей, ефективності, зручності використання та надійності захисту. Під час дослідження для порівняння було обрано п’ять відомих антивірусних продуктів**:**

- Zillya! Total Security;
- Avast Free Antivirus;
- Windows Defender;
- Bitdefender Internet Security;
- Avira Internet Security.

При порівнянні було визначено основні критерії, за якими оцінювалася кожна програма:

- **ефективність виявлення загроз** – здатність антивірусу швидко і точно виявляти як звичайні, так і складні загрози.
- **зручність використання** – простота інтерфейсу, доступність налаштувань, а також швидкість виконання операцій.
- **захист системи** – рівень безпеки в реальному часі, якість сканування, а також додаткові інструменти для запобігання вірусним атакам.’

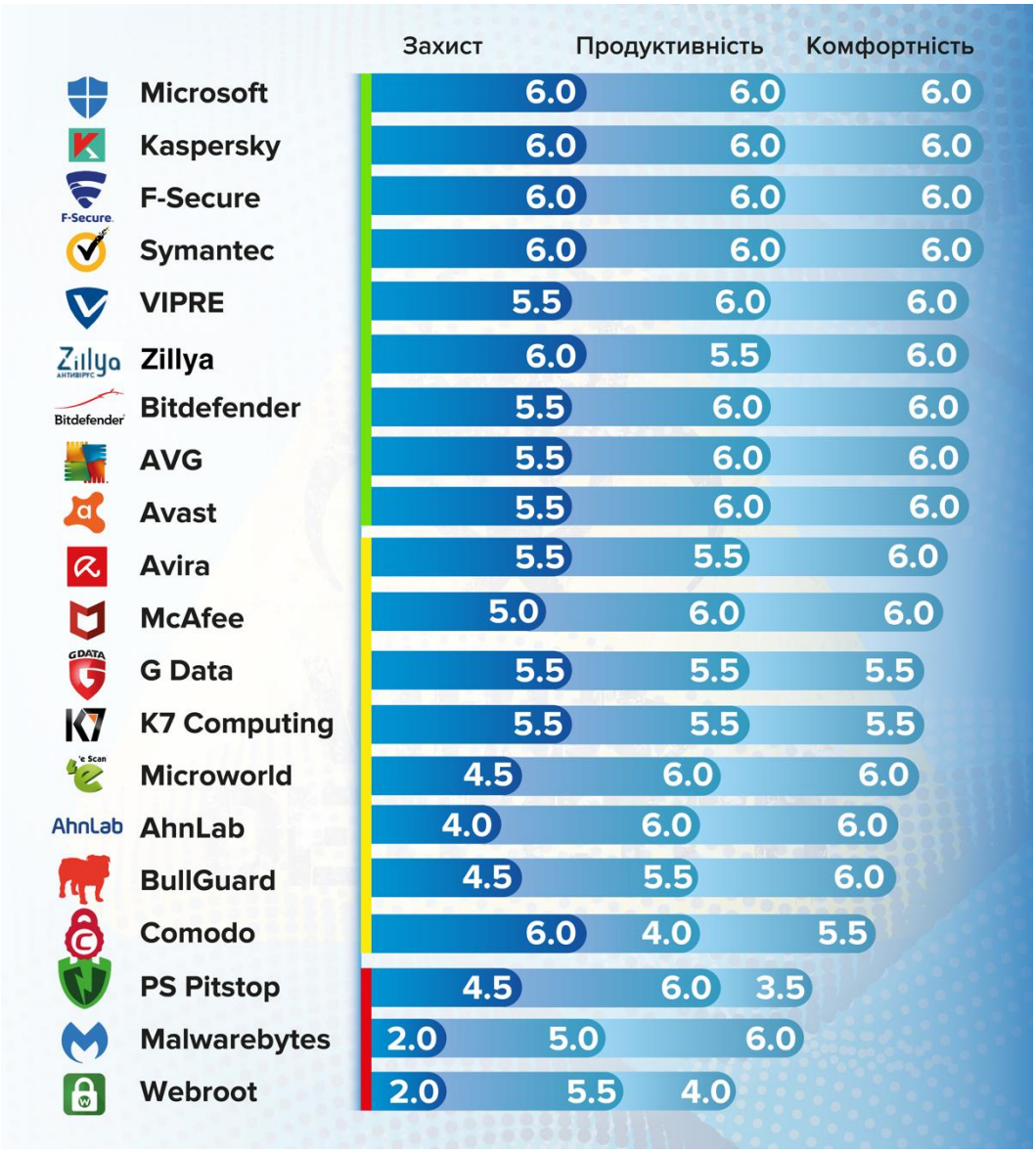

Рисунок 2.3 – Рейтинг антивірусних програм

1. Zillya! Total Security.

“Zillya! – український антивірус, розроблений у 2009 році компанією «Лабораторія Zillya!». Продукт орієнтований як на домашніх користувачів, так і на бізнес-сегмент. База даних програми налічує понад 15 мільйонів вірусів, що дозволяє виявляти загрози за короткий час” - [23].

Рисунок 2.4 - Zillya! Total Security

Функціонал антивірусу:

- аналіз програмного коду для виявлення вірусів;
- постійний моніторинг системи для виявлення потенційних загроз;
- захист від проникнення шкідливих програм через USB-накопичувачі.

Переваги:

- простий інтерфейс, що забезпечує зручну навігацію;
- швидке, повне та вибіркове сканування системи;
- велика база сигнатур, що постійно оновлюється.

Недоліки:

- після лікування заражених файлів можуть спостерігатися незначні порушення роботи системи Windows;
- існує ймовірність помилкового занесення чистих файлів у карантин;
- під час тестування не виявлено вірус типу **ntos.exe**, що свідчить про обмежені можливості при роботі зі складними загрозами.

2. Avast Free Antivirus.

“Avast є одним із найпопулярніших антивірусів у світі, розробленим чеською компанією AVG Technologies. Програму використовують понад 170 мільйонів користувачів по всьому світу” - [24].

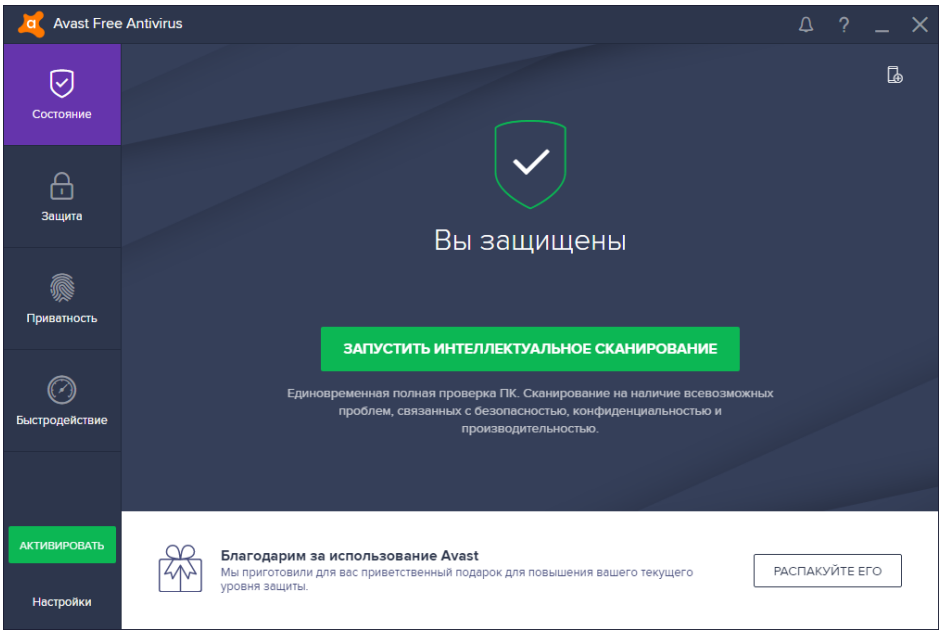

Рисунок 2.5 - Avast Free Antivirus

Функціонал Avast Free Antivirus:

- захист файлів та фото від програм-вимагачів;
- пошук вразливостей у Wi-Fi-мережі;
- моніторинг електронної пошти;
- перевірка системи на наявність загроз у реальному часі.

Переваги:

- доступна безкоштовна версія з базовим функціоналом;
- автоматичне оновлення баз даних;

- швидка перевірка всієї системи на віруси.

Недоліки:

- недостатньо ефективний захист від фішингових сайтів;
- складно працює з архівами та виконуваними файлами;
- при високому рівні безпеки антивірус значно навантажує систему;
- часто трапляються помилкові спрацьовування, коли чисті файли сприймаються як загрози.

3. Windows Defender.

“Windows Defender – вбудований антивірус для операційної системи Windows, розроблений корпорацією Microsoft. Програма автоматично активується під час встановлення Windows, забезпечуючи базовий рівень захисту” – [25].

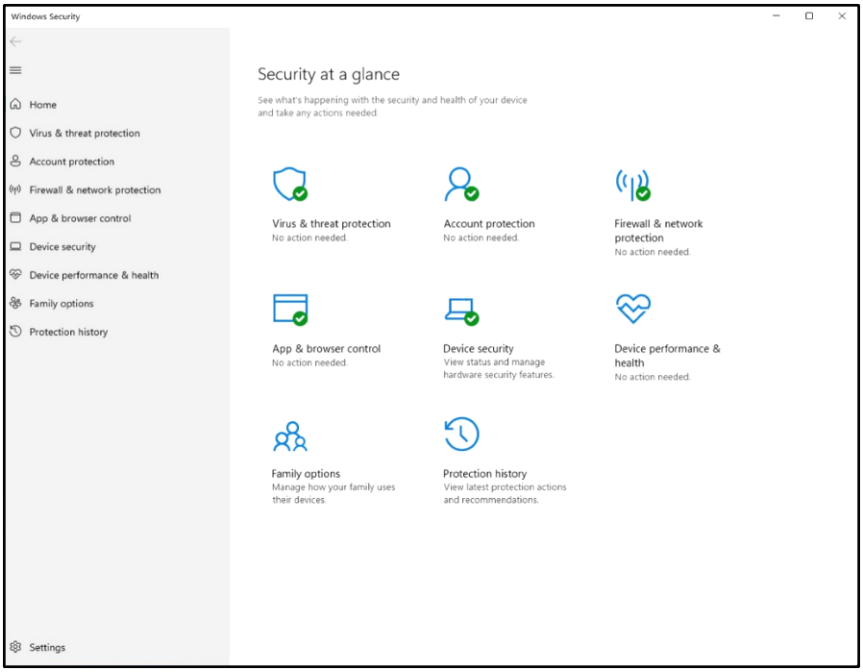

Рисунок 2.6 – Windows Defender

Функціонал Windows Defender:

- захист у реальному часі;
- глибоке сканування системи з можливістю налаштування графіку перевірок;

- автоматичне оновлення бази загроз.

Переваги:

- мінімальні вимоги до ресурсів системи;
- низький рівень помилкових спрацьовувань;
- відсутність реклами та прихованих зборів даних.

Недоліки:

- часто пропускає складні загрози;
- відсутність додаткових інструментів для підвищення рівня захисту.

4. Bitdefender Internet Security.

“Bitdefender – потужний антивірусний продукт, розроблений румунською компанією Bitdefender SRL. Програма надає багатошаровий захист для домашніх користувачів і бізнесу” – [26].

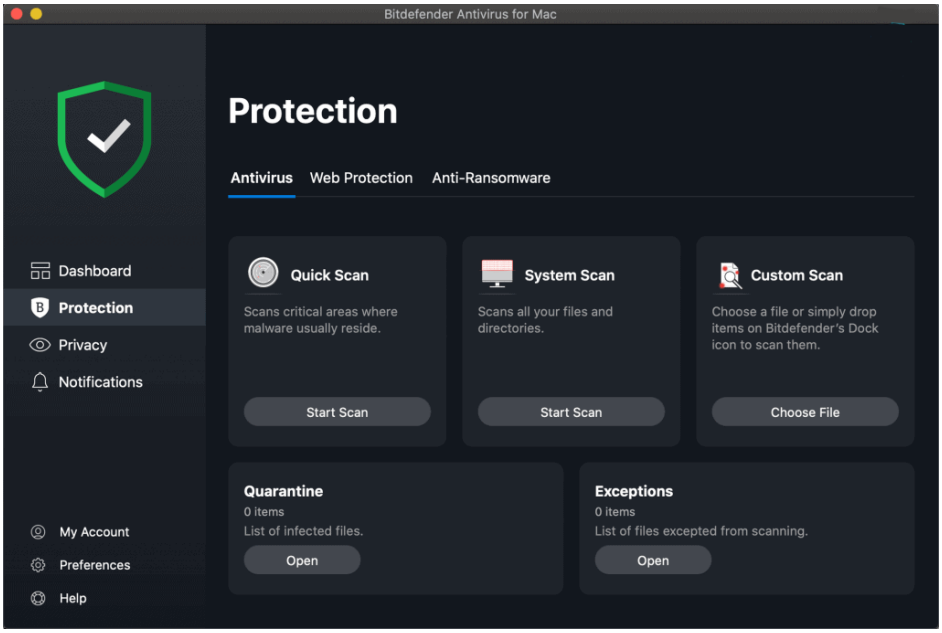

Рисунок 2.7. - Bitdefender Internet Security

Функціонал Bitdefender Internet Security:

- блокування вірусів, програм-вимагачів та шпигунського ПЗ;
- захист мережевих підключень (Wi-Fi та локальні мережі);
- батьківський контроль та захист конфіденційних даних.

Переваги:

- зручний інтерфейс із багатим функціоналом;
- високий рівень виявлення загроз;
- блокування фішингових сайтів та потенційно небезпечних URL-адрес.

Недоліки:

- більшість функцій доступні лише у платній версії;
- відсутність гнучких налаштувань для просунутих користувачів.

5. Avira Internet Security.

“Avira є одним із найкращих антивірусів для виявлення вірусів, програм-вимагачів та шпигунів” – [27].

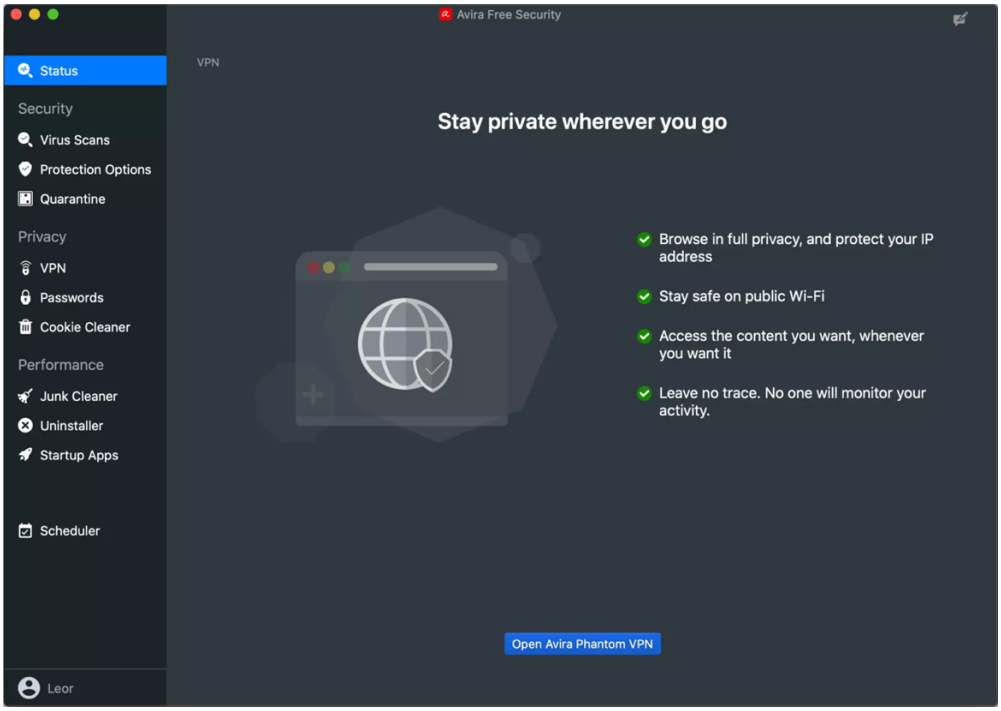

Рисунок 2.8 - Avira Internet Security

Функціонал Avira Internet Security:

- блокування вірусів, троянів та кейлогерів;
- захист від фішингових атак та підозрілої активності;
- швидке сканування жорсткого диску та файлів у хмарному сховищі.

Переваги:

- висока швидкість виявлення та блокування загроз;

- захист від витоків секретної інформації;
- сканування поштових клієнтів та веб-сервісів.

Недоліки:

- високе навантаження на систему, що ускладнює роботу на малопотужних ПК;
- потребує багато оперативної пам’яті;
- під час перевірки рекомендується закривати інші програми для уникнення збоїв.

Таблиця 2.1 - Порівняльний аналіз антивірусних програм

Антивірус	**Функціональні можливості**	**Ефективність**	**Зручність користування**	**Надійність захисту**
1	2	3	4	5
Zillya! Total Security	пропонує широкий набір інструментів, включно з функціями батьківського контролю, оптимізації системи та багаторівневого захисту.	виявилися менш ефективними у випадках новітніх шкідливих програм.	має зручний інтерфейс, але трохи застарілий дизайн.	має прийнятний рівень надійності, проте в складних загрозах поступається більш відомим конкурентам.
Avast Free Antivirus	забезпечує базовий захист із додатковими інструментами, як-от сканування	демонструє прийнятну ефективність для базового захисту, хоча в деяких	інтуїтивно зрозумілий, хоча користувачам можуть заважати	забезпечують базовий рівень надійності, але в деяких випадках

Продовження табл. 2.1

1	2	3	4	5
	Wi-Fi та захист веб-камери.	тестах поступається платним рішенням.	часті пропозиції придбати платну версію.	можуть давати хибні позитивні результати.
Windows Defender	нтегрований у Windows, має основні функції захисту від шкідливих програм і безкоштовно оновлюється разом із системою.	виявилися менш ефективними у випадках новітніх шкідливих програм.	має інтуїтивний інтерфейс і працює у фоновому режимі без значного навантаження на систему.	забезпечують базовий рівень надійності, але в деяких випадках можуть давати хибні позитивні результати.
Bitdefender Internet Security	забезпечує високий рівень захисту з додатковими функціями, такими як VPN, захист від фішингу та менеджер паролів.	демонструють найкращі результати у виявленні та нейтралізації загроз завдяки передовим технологіям поведінкового аналізу.	вирізняються сучасним дизайном і чіткою структурою налаштувань.	показали найвищу стабільність і низький рівень хибних спрацювань.
Avira Internet Security	пропонує комплексний захист, включаючи інструменти для	демонструють найкращі результати у виявленні та нейтралізації	вирізняються сучасним дизайном і чіткою	показали найвищу стабільність і низький рівень

Продовження табл. 2.1

1	2	3	4	5
	підвищення продуктивності системи та захисту конфіденційності.	загроз завдяки передовим технологіям поведінкового аналізу.	структурою налаштувань.	хибних спрацювань.

Підсумковий аналіз.

Серед досліджених антивірусних програм **Bitdefender Internet Security** та **Avira Internet Security** є лідерами за сукупністю функціональних можливостей, ефективності та надійності. Вони підходять для користувачів, які потребують максимального захисту і готові інвестувати в платні рішення. **Windows Defender** є оптимальним варіантом для базового безкоштовного захисту з мінімальним втручанням у роботу системи. **Avast Free Antivirus** підходить тим, хто шукає безкоштовне рішення з розширеними функціями, але готовий миритися з рекламними пропозиціями. **Zillya! Total Security** є гідним вибором для користувачів, які підтримують локальні розробки, хоча продукт має потенціал для вдосконалення.

Результати дослідження свідчать, що вибір антивірусної програми залежить від потреб користувача: для максимального захисту та функціональності доцільно обрати преміальні рішення, а для базового захисту вистачить безкоштовних альтернатив.

РОЗДІЛ 3
РОЗРОБКА АНТИВІРУСНОЇ ПРОГРАМИ, ЯК МЕТОДУ ЗАХИСТУ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

3.1 Постановка задачі, призначення та вимоги до програмного засобу

Програмний засіб **«My_Antivirus»** розробляється з метою **виявлення, ізоляції та ліквідації комп'ютерних загроз** для забезпечення комплексного захисту інформаційних систем. Комп'ютерні віруси, приховані у файлах, програмах, документах або системних компонентах, можуть призводити до пошкодження даних, втрати інформації, порушення конфіденційності та некоректної роботи операційної системи. Програма буде спрямована на запобігання подібним загрозам, забезпечуючи стабільність роботи системи **та** надійність збереження інформації**.**

В умовах зростання кількості вірусних атак та шкідливих програм **необхідно створити інструмент, що забезпечить**:

- високий рівень ефективності при скануванні системи на предмет загроз;
- інтуїтивно зрозуміле управління;
- можливість швидкого реагування на потенційні небезпеки.

Основні задачі програмного засобу:

- виявлення загроз шляхом аналізу файлів, програм та системних компонентів на наявність шкідливого коду;
- сканування системи та окремих файлів у різних режимах для максимально ефективного виявлення вірусів;
- сканування URL-адрес для перевірки веб-ресурсів на наявність потенційних загроз, таких як фішингові сайти або шкідливі посилання;
- ізоляція виявлених загроз у спеціальному середовищі (карантині) для подальшого аналізу або видалення;

- автоматичне видалення шкідливих об’єктів, що становлять загрозу для системи;
- оновлення антивірусної бази для виявлення новітніх вірусів і шкідливого програмного забезпечення;
- надання користувачеві звітності про результати сканування, виявлені загрози та виконані дії;
- гарантування простоти використання завдяки продуманому інтерфейсу, зрозумілому для користувача будь-якого рівня.

Вимоги до програмного засобу.

Розробка програмного продукту «**My_Antivirus**» передбачає реалізацію наступних функціональних та нефункціональних вимог:

1. Функціональні вимоги:

- **вибіркове сканування файлів** - користувач має можливість вибрати окремі файли, папки або програми для перевірки на наявність вірусів і загроз;
- **повне сканування системи** - передбачено можливість виконання повного аналізу всієї операційної системи, включно з прихованими файлами та системними процесами;
- **сканування URL-адрес** - передбачено можливість сканування URL адрес дозволить аналізувати веб-посилання на предмет потенційних загроз у режимі реального часу, допомагаючи уникнути доступу до небезпечних веб-ресурсів;
- **карантин** - передбачено створення ізольованого середовища для підозрілих файлів. Користувач може:
  - тимчасово перемістити шкідливі файли в карантин;
  - переглянути їх статус та інформацію;
  - прийняти рішення про видалення чи відновлення.
- **антивірусна база** - програма використовує **сигнатурний метод** виявлення загроз, що базується на відомих зразках вірусів.
- **зручний інтерфейс**:
  - інтуїтивно зрозумілий для користувачів будь-якого рівня;

– містить ключові функції на головному екрані програми;

– підтримує **денний** та **нічний режим** для комфорту очей;

– підтримує **дві мови**: українська та англійська;

– забезпечує **швидкий доступ** до режимів сканування, карантину та звітів.

- **звітність** - програма формує докладні звіти про результати сканування, час виконання, виявлені загрози та виконані дії.

2. Нефункціональні вимоги:

- **продуктивність** - програма повинна працювати швидко, не створюючи значного навантаження на ресурси комп'ютера, особливо при фоновому скануванні;
- **сумісність** - антивірус повинен працювати на популярних операційних системах, таких як Windows, MacOS, Linux;
- **безпека** - програмний продукт має гарантувати захист конфіденційності користувачів та їх даних, не збираючи особисту інформацію без дозволу;
- **масштабованість** - передбачено можливість розширення функціоналу в майбутньому для адаптації до нових загроз і потреб користувача.

Вимоги до інтерфейсу.

Для забезпечення максимальної зручності користування, інтерфейс повинен:

- мати чітко структуроване меню, що забезпечує швидкий доступ до функцій (сканування, карантин, налаштування);
- відображати поточний статус системи та результати сканування;
- підтримувати темну та світлу теми для адаптації до умов роботи користувача;
- забезпечувати наочність результатів сканування у вигляді звітів, де буде вказано: час виконання перевірки, кількість перевірених файлів, виявлені загрози та дії, що були виконані над ними;

- містити **візуальні індикатори статусу безпеки** (наприклад, зелений колір – система безпечна, червоний – загроза виявлена).

Технічні особливості розробки:

- **мова програмування**: для розробки програмного засобу планується використання сучасної мови програмування **Python** через його потужні бібліотеки для роботи з файлами та безпекою;
- **середовище розробки**: **Visual Studio Code**, як універсальне середовище для створення програм та веб-додатків;
- **обробка сканування**: Для підвищення точності та актуальності аналізу файлів, система інтегрована з **VirusTotal API**. Це дозволяє перевіряти файли на основі хмарної бази даних, що містить інформацію про шкідливі об'єкти від декількох антивірусних рішень.

Таким чином, програмний засіб «**My_Antivirus**» буде універсальним інструментом для **виявлення, ізоляції та видалення загроз**, при цьому забезпечуючи зручність у використанні, високу продуктивність та стабільність роботи системи.

3.2 Вибір моделі розробки програмного засобу «My_Antivirus»

“Метод для сканування файлів через інтеграцію з VirusTotal API є виправданим, зручним і ефективним завдяки низці переваг, які поєднують у собі функціональність, надійність та простоту реалізації. VirusTotal має широкий спектр перевірок завдяки багатьом антивірусним двигунам” – [22].

“VirusTotal API використовує понад **70 антивірусних двигунів** (антивірусів) для аналізу файлів. Це забезпечує високу точність і надійність сканування” – [22].

“Один локальний антивірус може не виявити новітні або маловідомі загрози, але комбінована перевірка різними антивірусами значно підвищує шанси виявлення” – [22].

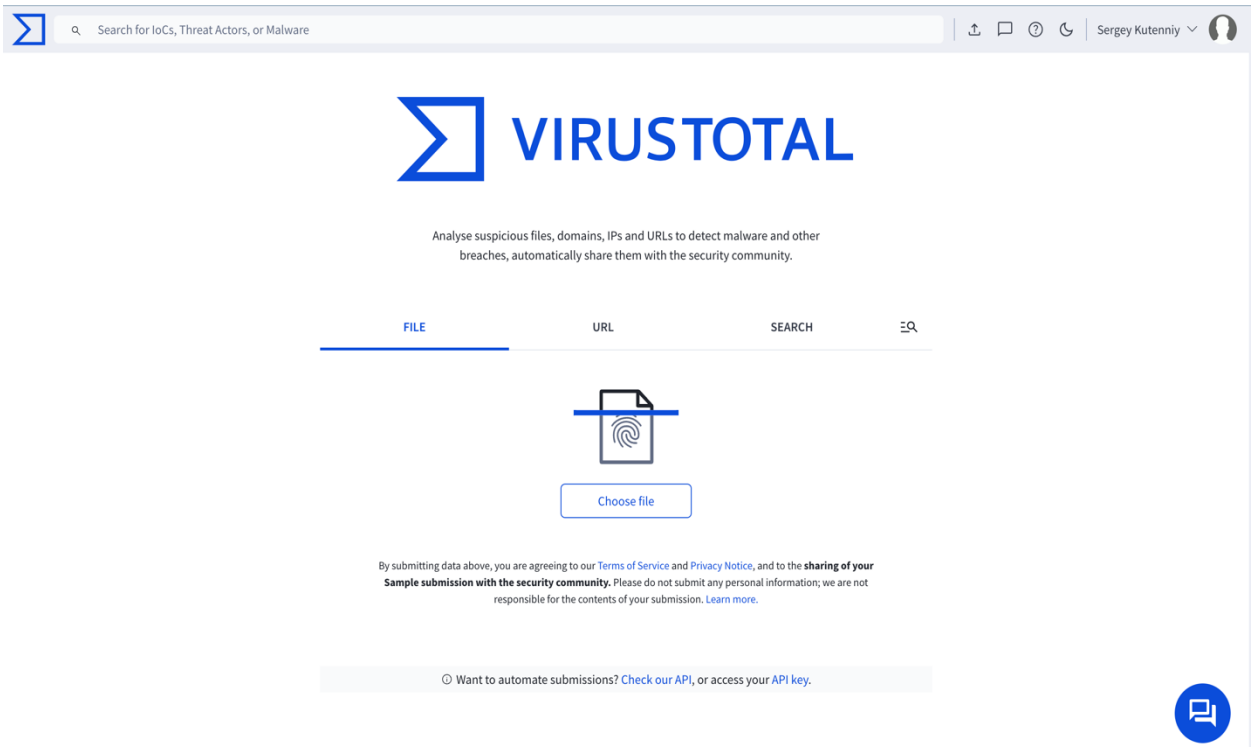

Рисунок 3.1 – Платформа Virustotal

1. Швидкість та доступність сканування.

“Сканування файлів через API VirusTotal не потребує локальних ресурсів користувача, оскільки перевірка відбувається на серверах VirusTotal” – [22].

- **перевага**: мінімальне навантаження на центральний процесор (CPU) та оперативну пам'ять (RAM) локального пристрою;
- **зручність**: сканування може відбуватися у фоновому режимі або під час роботи з іншими файлами;
- **реальна ситуація**: користувачі на слабких пристроях (ноутбуках чи нетбуках) можуть сканувати файли, папки чи URL без ризику уповільнення системи.

2. Легкість інтеграції у програмне середовище.

“VirusTotal API є документованим і простим у використанні, що дозволяє легко інтегрувати його у будь-який додаток чи програмне забезпечення” – [22].

- **сучасний RESTful API**: відправка запитів через HTTP-протокол за допомогою бібліотеки **requests** у Python робить інтеграцію простою;
- **JSON-відповіді**: результати сканування надходять у зручному форматі JSON, який легко парсити і відображати користувачу;
- **кодова реалізація**: кілька рядків коду дозволяють надсилати файл на перевірку та отримувати детальний звіт.

3. Оновлювана база даних загроз.

“VirusTotal постійно оновлює свої бази даних і використовує найсвіжіші сигнатури для виявлення шкідливих програм” – [22].

- **перевага**: користувачам не потрібно оновлювати антивірусні бази вручну. Всі оновлення відбуваються на стороні сервісу VirusTotal;
- **зручність**: програма, що використовує VirusTotal API, завжди має доступ до найновіших даних про віруси, трояни, бекдори, руткіти та інші загрози;
- **реальна ситуація**: швидке виявлення загроз “нульового дня” (нові віруси, які ще не розпізнаються традиційними антивірусами).

4. Кросплатформеність і віддалене використання.

“VirusTotal API працює через **HTTP-запити**, що робить його незалежним від платформи чи операційної системи” – [22].

- **перевага**: API можна використовувати у програмі для Windows, Linux, macOS або навіть у мобільних додатках для iOS та Android;
- **гнучкість**: розробники можуть інтегрувати VirusTotal API у веб-застосунки, серверні скрипти або мобільні програми.

5. Зручність звітності.

“VirusTotal API надає **детальні звіти про кожен файл або URL**” – [22]:

- місцезнаходження файлу в системі користувача;
- назви загроз (наприклад: **"Trojan.Generic"**, **"Malware.Packed", eicar.com**);
- які антивірусні системи виявили загрозу;
- кількість спрацювань;

- статистику аналізу (безпечний/небезпечний).

Переваги:

- легко обробляти результати та надавати користувачу у зрозумілому форматі;
- звіти можна інтегрувати у програмний інтерфейс, що підвищує зручність користування.

6. Відсутність необхідності у підтримці власного антивірусного ядра.

“Розробка та підтримка власного антивірусного ядра є **складною,** дорогою і часозатратною задачею. Використання VirusTotal API усуває цю необхідність” – [22].

- **перевага**: розробникам не потрібно оновлювати антивірусні сигнатури чи алгоритми сканування;
- **фокус на додатку**: час можна витратити на покращення функціоналу програми, її інтерфейсу та загальної продуктивності.

7. Економія часу та ресурсів.

“Сканування через VirusTotal API є ефективним і швидким завдяки централізованій інфраструктурі” – [22].

- **економія часу**: аналіз великих файлів або складних загроз виконується серверною інфраструктурою VirusTotal;
- **економія ресурсів**: локальний комп'ютер або сервер не перевантажується під час сканування.

8. Безпека і довіра.

“VirusTotal належить компанії Google, що підвищує рівень довіри до сервісу. Дані обробляються безпечно та конфіденційно” – [22].

- **перевага**: інформація про перевірені файли доступна лише з обліковим ключем (API Key), а приватність користувача захищена.

“Метод для сканування файлів через інтеграцію з **VirusTotal API** дозволяє розробникам зосередитися на головному – покращенні функціоналу програми, надаючи користувачу надійний і швидкий спосіб перевірки файлів на наявність загроз” – [22].

3.3 Опис проекту «My_Antivirus»

Програма, “**My_Antivirus”**, є графічним інтерфейсом користувача (GUI) для антивірусного програмного забезпечення, розробленого з метою сканування файлів, аналізу загроз і взаємодії з VirusTotal API для перевірки безпеки об'єктів.

Цей інтерфейс було створено з використанням бібліотеки **PyQt5**, яка забезпечує гнучкий та зручний графічний інтерфейс. Програма орієнтована на виявлення загроз у локальних файлах, url-адресах, виконання їхнього аналізу та надання результатів у зрозумілому форматі. Окрім локального сканування, програма підтримує інтеграцію з VirusTotal API, що дозволяє перевіряти об'єкти на наявність вірусів за допомогою популярного онлайн-сервісу.

Рисунок 3.1 – Графічний інтерфейс програми “My_Antivirus”

3.3.1 Основні функціональні можливості:

1. **Графічний інтерфейс користувача (GUI)**:

- програма використовує **PyQt5** для створення інтуїтивно зрозумілого інтерфейсу;

- інтерфейс містить кнопки, текстові поля, та інші елементи для роботи з файлами та результатами аналізу.

2. **Вибір файлів для сканування**:

- користувач може обрати один або кілька файлів, папку або url-адресу для подальшого аналізу;
- файли додаються у список для обробки, який відображається у графічному інтерфейсі.

3. **Сканування локальних файлів**:

- програма аналізує локальні файли за допомогою простих перевірок (наприклад, хешування файлів);
- обчислюється **SHA256-хеш** файлу, який використовується для подальшої перевірки через VirusTotal API.

4. **Інтеграція з VirusTotal API**:

- за допомогою VirusTotal API програма відправляє **SHA256-хеш** файлу або сам файл на сервер для аналізу;
- отримані результати містять інформацію про загрозу, кількість позитивних спрацювань та назви антивірусів, які виявили загрозу.

5. **Обробка відповідей та вивід результатів**:

- після перевірки файлу програма обробляє отриману відповідь від VirusTotal у форматі JSON;
- виводиться загальна статистика (кількість сканерів, позитивні результати, статус безпеки файлу).

6. **Сканування URL-адрес**:

- програма підтримує можливість перевірки URL-адрес на наявність шкідливого вмісту;
- виконується POST-запит до VirusTotal API, після чого результати аналізу відображаються користувачу.

7. **Повідомлення про помилки**.

Програма містить обробники помилок для випадків, коли:

- не обрано жодного файлу;

- виникають проблеми з підключенням до VirusTotal API;
- забагато запитів «error 429»;
- та ін.

8. Розділ карантину:

- файли переміщуються в розділ карантину для подальшої ізоляції;
- файли можуть бути видалені з системи користувача або відновлені в вихідне місцерозташування.

9. Розділ налаштувань програми:

- можливість обрати світлу чи темну тему програми;
- можливість обрати мову програми (українська, англійська).

3.3.2 Архітектура програми:

1. **Інтерфейс користувача**:

- створюється клас MainWindow, що успадковується від QMainWindow (клас PyQt5);
- у цьому класі визначаються всі елементи GUI: кнопки, текстові поля, таблиці тощо;
- **основні функції GUI:**

– кнопка для вибору файлів;
– кнопка для вибору папок;
– кнопка сканування URL-адрес;
– кнопка карантину;
– кнопка налаштувань програми;
– кнопка виходу з програми;
– поле для відображення результатів перевірки;
– індикатор виконання сканування (progress-bar).

2. **Сканування файлів**:

- після вибору файлів запускається функція для обчислення **SHA256-хешу** кожного файлу;

- отриманий хеш використовується для перевірки файлу через VirusTotal API.

3. **Інтеграція з VirusTotal API**:

- відправка файлу або хешу здійснюється через **HTTP-запит** із використанням бібліотеки **requests**;
- URL для запиту:
  - для файлів: https://www.virustotal.com/api/v3/files;
  - для хешів: https://www.virustotal.com/api/v3/files/{id};
  - для URL: https://www.virustotal.com/api/v3/urls.
- Для автентифікації використовується заголовок **x-apikey**.

4. **Обробка JSON-відповіді**:

- отримані відповіді розбираються за допомогою бібліотеки json;
- основні поля у відповіді:
  - **data.attributes.stats** - статистика сканування;
  - **data.attributes.results** - детальні результати сканування (назви антивірусів і їхні висновки).
- результати відображаються у відповідному полі.

5. **Обробка URL-адрес**:

- URL-адреси відправляються на перевірку через POST-запит;
- після аналізу результати відображаються у GUI.

6. **Обробка помилок**:

- перевіряється правильність API-ключа;
- обробляються помилки мережевого підключення;
- користувач отримує повідомлення про можливі проблеми.

3.3.3 Огляд використаних бібліотек, технологій та інструментів:

1. “Python — це високорівнева мова програмування загального призначення, яка відзначається простим синтаксисом, читабельністю та гнучкістю. Вона підтримує кілька парадигм програмування, включаючи об'єктно-орієнтоване програмування (ООП), функціональне програмування та

процедурне програмування. Python є інтерпретованою мовою, що означає виконання коду рядок за рядком, що полегшує розробку, тестування та налагодження" – [14].

2. "PyQt5 — це набір інструментів для створення графічного інтерфейсу користувача (GUI) за допомогою мови програмування Python. Цей модуль є обгорткою для фреймворку Qt, який популярний для кросплатформної розробки інтерфейсів додатків – [14].

Функції та використання:

- **створення GUI** — підтримка основних компонентів інтерфейсу, таких як кнопки, поля введення, меню, вікна повідомлень;
- **обробка подій** — сигнально-слотовий механізм для взаємодії між елементами інтерфейсу;
- **гнучка кастомізація** — можливість створення унікального та зручного інтерфейсу завдяки стилям та властивостям віджетів;
- **кросплатформність** — можливість запускати додатки на Windows, macOS та Linux без зміни коду.

Основні елементи:

- QmainWindow - головне вікно програми;
- QpushButton - кнопка для виконання дії;
- QTextEdit/QLineEdit - текстові поля для вводу/виводу;
- QtableWidget - таблиці для відображення результатів;
- QMessageBox - відображення інформаційних, попереджувальних чи помилкових діалогових вікон.

Роль у програмі:

PyQt5 використовується для створення інтерфейсу антивірусного програмного забезпечення, де:

- користувач може завантажити файл;
- натиснути кнопку для сканування;
- переглянути результати аналізу в зручному форматі.

3. “**Requests** — це популярна бібліотека для роботи з HTTP-запитами у Python. Вона дозволяє надсилати HTTP-запити та отримувати відповіді з вебсервісів чи API” – [14].

Функції та використання:

- GET-запити — отримання даних з вебресурсу або API;
- POST-запити — надсилання даних на сервер для обробки;
- JSON-підтримка — просте декодування відповідей у форматі JSON;
- помилки та статуси — вбудована обробка HTTP-статусів для контролю результатів запитів.

Роль у програмі:

- бібліотека requests забезпечує взаємодію програми з VirusTotal API для перевірки файлів на наявність вірусів;
- виконує HTTP-запити для завантаження файлів та отримання результатів сканування у форматі JSON.

4. Інтеграція з VirusTotal API:

- **отримання ключа API**:
  - зареєструвати обліковий запис на офіційному сайті VirusTotal;
  - отримати безкоштовний API-ключ з особистого кабінету.
- **сканування файлів**:
  - надсилання файлу на сервіс для аналізу;
  - отримання ідентифікатора файлу (file_id) для подальшого запиту результатів.
- **отримання звітів**:
  - надсилання запиту з file_id для отримання результатів сканування.

Роль у програмі:

- сканування файлів — користувач може завантажити файл для перевірки;
- звіт про загрози — програма отримує результати аналізу, які можна відобразити у зрозумілому форматі;

- інтеграція з інтерфейсом (PyQt5) — результати сканування відображаються у вікні програми, в текстовому полі.

5. "JSON — це вбудована бібліотека Python, яка використовується для роботи з даними у форматі **JSON (JavaScript Object Notation)**. JSON є легким, текстовим форматом обміну даними, який використовується для зберігання, передачі та обміну структурованими даними між клієнтськими та серверними додатками" – [14].

Основні функції бібліотеки JSON:

- серіалізація (перетворення об'єктів Python у JSON-рядок) — використовується для передачі даних через мережу або збереження у файли;
- десеріалізація (перетворення JSON-рядка у об'єкт Python) — використовується для обробки даних, отриманих з API або файлів;
- читання та запис JSON у файли — дозволяє працювати з файлами, які містять дані у форматі JSON.

Використання JSON у програмі для інтеграції з **VirusTotal API дозволяє:**

- розпаковувати відповіді сервера у форматі JSON — кожен HTTP-запит до VirusTotal повертає JSON-відповідь, яку можна обробити за допомогою json.loads();
- зберігати дані у JSON-файли — результати перевірки файлів можуть бути збережені для подальшого аналізу.

6. "Модуль **OS** — це вбудована бібліотека Python, яка надає функції для взаємодії з операційною системою. Він дозволяє виконувати різноманітні операції на файловій системі, працювати зі шляхами, змінними середовища та виконувати системні команди" – [14].

Основні функції модуля OS:

- робота з файловою системою;
- отримання інформації про файли та шляхи;
- системні команди;
- робота з процесами;

- поточний робочий каталог;
- перевірка операційної системи.

Ключові особливості:

- модуль OS є платформонезалежним, але деякі функції працюють лише на конкретних ОС;
- для складнішої роботи з файлами рекомендується використовувати модуль **shutil** або бібліотеку **pathlib**.

7. "Модуль **shutil** — це вбудована бібліотека Python, яка надає інструменти для виконання високорівневих операцій із файлами та папками. Цей модуль особливо корисний для автоматизації роботи з файлами та каталогами. Його функції часто використовуються у сценаріях резервного копіювання, організації файлів і створення архівів." – [14].

Основні функції модуля shutil:

- копіювання файлів і папок;
- переміщення, перейменовування, видалення файлів і папок;
- отримання інформації про використання диску;
- створення архівів;
- управління дисковим простором;

8. "Модуль **datetime** надає класи для роботи з датами, часом і часовими інтервалами. Його використовують для отримання поточного часу, обчислення різниці між датами, форматування виводу тощо" – [14].

Основні характеристики модуля datetime:

- представлення часу;
- гнучкість форматування;
- обчислення часу;
- підтримка універсального часу (UTC);
- використання UNIX-міток.

Застосувується модуль datetime в багатьох практичних задачах:

- облік часу та дат — наприклад, ведення журналу подій або управління розкладом.

- таймінг і планування — автоматизація завдань, які виконуються у певний момент або через заданий проміжок часу.
- фінансові обчислення — аналіз часових інтервалів для оцінки прибутків, відсотків чи виконання контрактів.
- глобалізація — облік часових зон і коректне відображення часу для користувачів з різних регіонів.

9. “Visual Studio Code (VS Code) — це безкоштовний, кросплатформенний редактор коду, розроблений компанією Microsoft. Він призначений для створення, налагодження та управління програмними проектами, забезпечуючи розробників інструментами, які значно підвищують продуктивність. VS Code підтримує широкий спектр мов програмування та має розширену екосистему плагінів” – [14].

Основні характеристики VS Code:

- **легка вага і продуктивність:**

– VS Code працює швидко, навіть на комп'ютерах із середніми технічними характеристиками;

– це ідеальний вибір для невеликих проєктів, а також для великих, комплексних програмних систем.

- **кросплатформенність:**

– редактор доступний для Windows**,** macOS та Linux, що дозволяє працювати з ним на будь-якій операційній системі.

- **підтримка багатьох мов:**

– вбудована підтримка популярних мов програмування, таких як Python, JavaScript, TypeScript, Java, C++, C#, PHP тощо;

– додаткова підтримка мов через розширення.

- **вбудований термінал:**

– можливість запускати командний рядок безпосередньо в редакторі;

– зручність роботи з термінальними командами без необхідності перемикатися між вікнами.

- **інтелектуальна підказка (IntelliSense):**

– автодоповнення синтаксису, функцій, змінних і параметрів;

– контекстна підказка для прискорення написання коду.

- **налагодження (Debugging):**

– вбудовані інструменти для запуску й налагодження програм безпосередньо з редактора;

– підтримка точок зупинки, перевірки змінних та інших засобів діагностики.

- **плагіни та розширення:**

– широка бібліотека розширень дозволяє додавати нові функції, такі як підтримка мов, інтеграція з інструментами CI/CD, управління проєктами тощо;

– популярні розширення: Python Extension Pack**,** Prettier**,** ESLint**,** Docker**,** GitLens**,** Live Server.

- **інтеграція з Git:**

– VS Code підтримує Git із коробки, дозволяючи виконувати команди на кшталт git commit, pull, push тощо;

– інтуїтивний інтерфейс для управління репозиторіями та аналізу змін.

Сфери застосування VS Code:

- **розробка програмного забезпечення -** підтримка багатьох мов і технологій робить VS Code ідеальним для створення десктопних, мобільних та вебдодатків.
- **веброзробка -** інструменти, такі як Live Server та CSS IntelliSense, дозволяють швидко працювати з HTML, CSS і JavaScript.
- **робота з даними та наукові обчислення -** плагіни для Python, Jupyter та R дозволяють виконувати обчислення, аналіз і візуалізацію.
- **DevOps та CI/CD -** інтеграція з Docker, Kubernetes і хмарними платформами, такими як Azure та AWS.
- **управління версіями -** просте підключення до GitHub, Bitbucket, GitLab або інших Git-систем.

Переваги VS Code:

- швидкість і легкість використання;
- широка екосистема розширень;
- інтеграція з популярними інструментами, такими як Git, Docker, Virtual Studio App Center;
- безкоштовний доступ.

Недоліки VS Code:

- для роботи з великими проєктами можуть бути потрібні додаткові налаштування;
- інколи виникає залежність від плагінів, що може вплинути на продуктивність;
- менш потужний, ніж повноцінна IDE (наприклад, Visual Studio або IntelliJ IDEA), для складних корпоративних додатків.

3.4 Програмування програмного продукту «My_Antivirus»

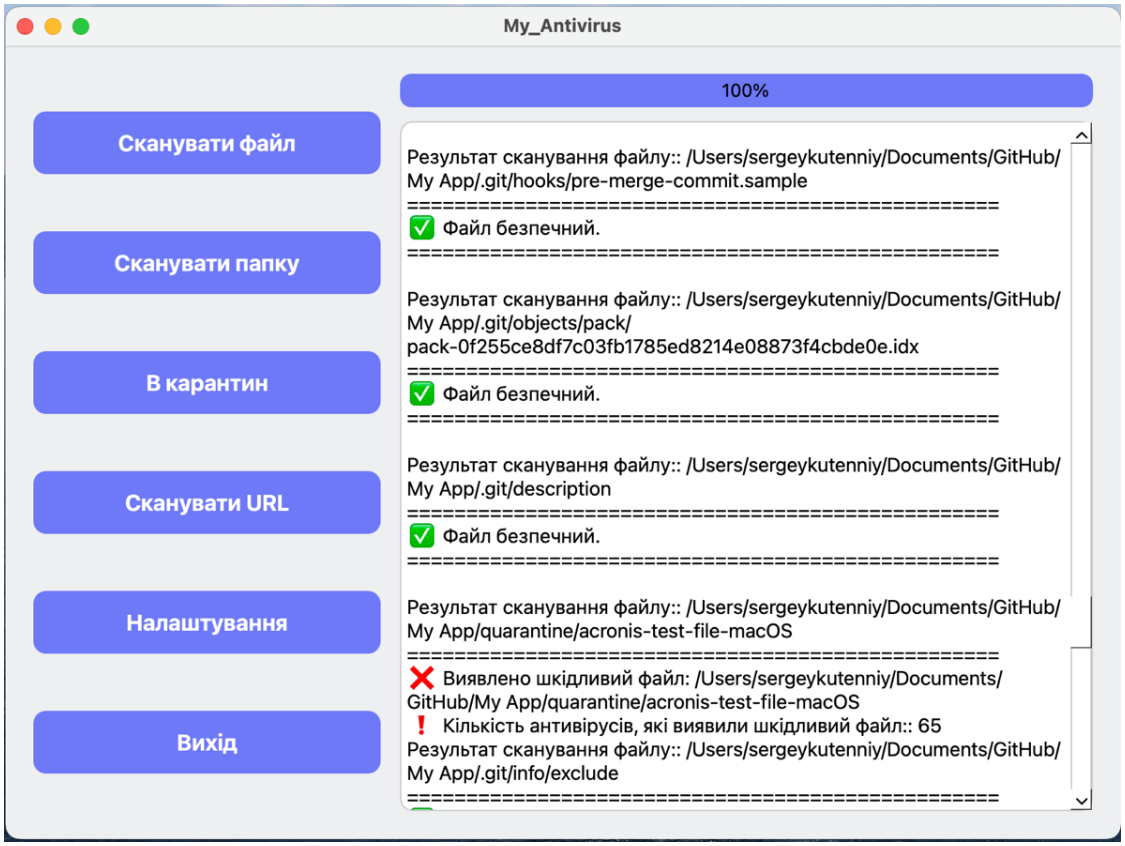

Рисунок 3.2 – Графічний інтерфейс програми “My_Antivirus”

3.4.1 Кроки які використовувались для створення “My_Antivirus”.

Основна логіка запуску програми:

- встановлюється базова точка входу в програму;
- створюється екземпляр класу QApplication для запуску графічного інтерфейсу;
- ініціалізується основне вікно програми через mainwindow.MainWindow();
- запускається основний цикл програми через sys.exit(app.exec_()).

main.py

```
from PyQt5.QtWidgets import QApplication
from windows import mainwindow

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    mw = mainwindow.MainWindow()
    sys.exit(app.exec_())


```

Рисунок 3.3 – Основний файл запуску програми main.py

Графічний інтерфейс.

Файл mainwindow.py відповідає за візуалізацію та взаємодію користувача з програмою.

Основні компоненти:

- MainWindow - головне вікно програми;
- SettingsDialog - вікно для налаштувань мови та теми;
- QuarantineWindow - для перегляду карантину;
- FileScanThread - окремий потік для сканування файлів.

Функціонал:

- панель управління:

– кнопки: "Сканувати файл", "Сканувати папку", "Сканувати URL", "Карантин", "Налаштування", "Вихід", "Видалити файл", "Відновити файл";

– реалізація через QPushButton, кожна кнопка прив'язана до відповідної функції.

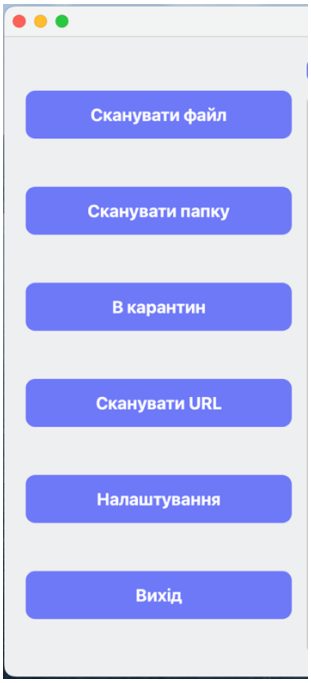

Рисунок 3.4 – Панель управління “My_Antivirus”

- **відображення результатів**:

– індикатор сканування (QProgressBar);

– текстове поле (QTextEdit) для результатів сканування.

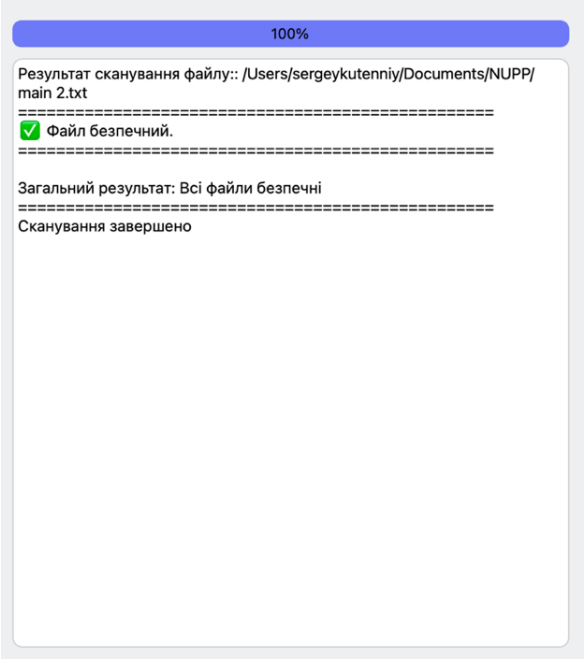

Рисунок 3.5 – Панель відображення результатів та індикатору сканування “My_Antivirus”

- **сканування файлів та URL** - інтеграція з модулем **virustotal** для перевірки файлів та URL на наявність шкідливого ПЗ з діалоговим вікном для введення URL.

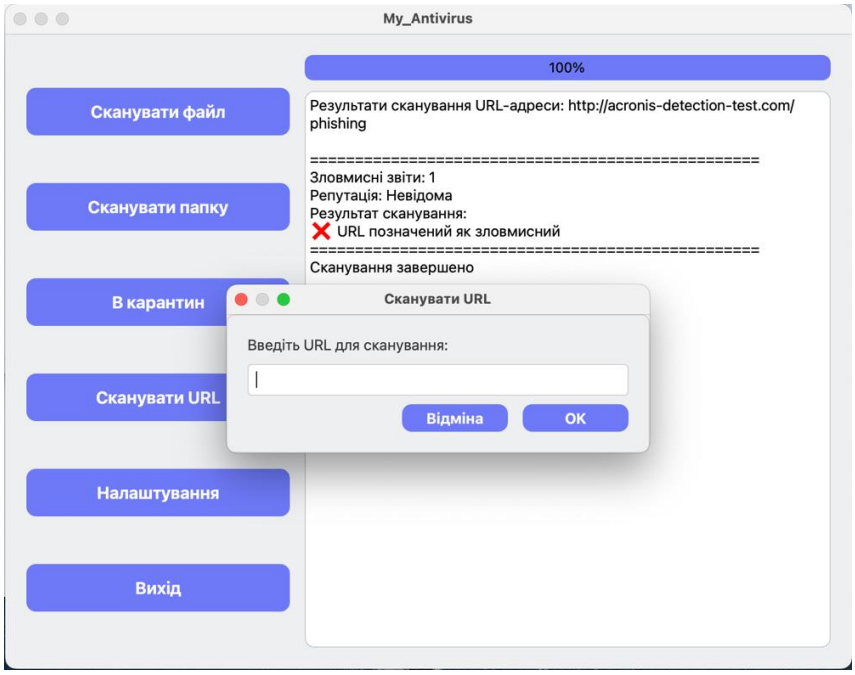

Рисунок 3.6 – Сканування URL

- **карантин**:

– логування у файлів у **quarantine_log.json;**

– опції для видалення або відновлення файлів;

Рисунок 3.7 - Карантин

- **потоки** - для асинхронного сканування використовується FileScanThread;
- **налаштування теми та мови**:

– зміна мови (Українська/English) через переводи текстів;

– теми (світла/темна) змінюють кольорові схеми всіх компонентів.

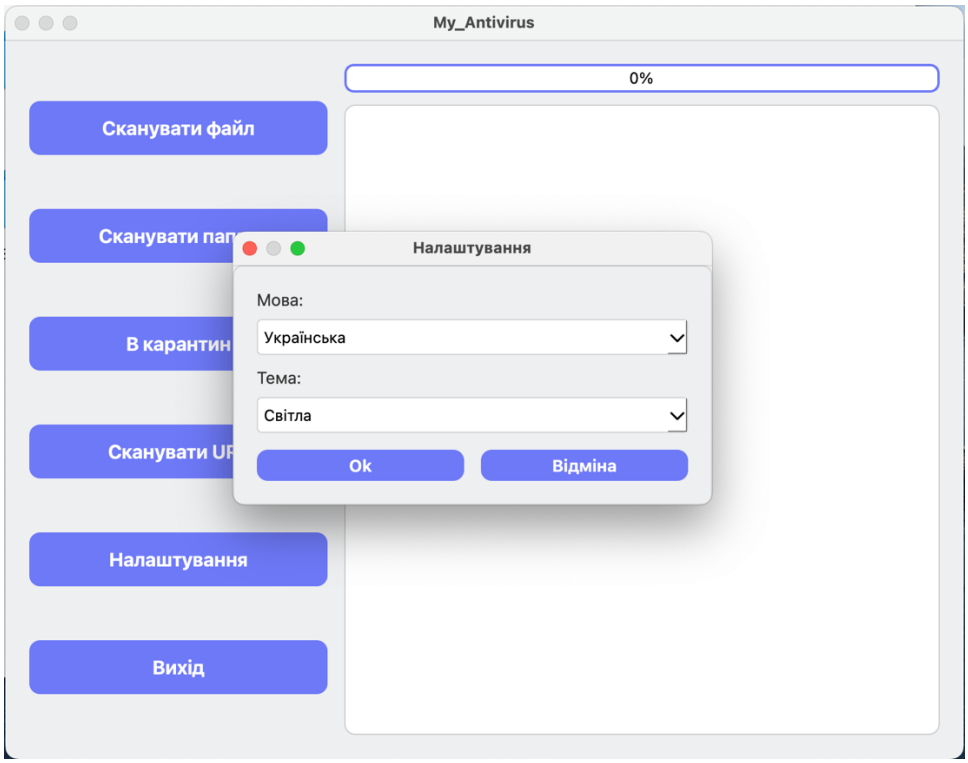

Рисунок 3.8 – Вікно налаштувань програми “My_Antivirus”

Файл **virustotal.py** містить логіку взаємодії з **API VirusTotal**.

Основні функції:

- **upload_file(path)**:

– завантажує файл на сервер VirusTotal для перевірки;

– отримує ID аналізу та результат;

– аналізує відповіді (кількість "malicious", "harmless", "suspicious", "undetected").

- **parse_analysis_results(results)**:

– форматує дані аналізу в зручний для відображення формат.

- **scan_url(url)**:

– відправляє URL для перевірки;

– отримує ID аналізу та деталі результатів (наприклад, скільки антивірусів позначили URL як небезпечний).

Рисунок 3.9 - Логіка взаємодії з **API VirusTotal**

Можливі покращення:

- реалізувати кешування результатів сканування;
- додати можливість працювати з іншими API або локальними базами сигнатур;
- можливість фонового сканування системи.

3.4.2 Опис створення програмного продукту «My_Antivirus». Основна логіка запуску програми.

Файл **main.py** є точкою входу в програму. У ньому визначено мінімальний обсяг коду, необхідного для запуску графічного інтерфейсу та ініціалізації головного вікна.

Імпорт модулів:

- PyQt5.QtWidgets - для створення графічних інтерфейсів;
- mainwindow - модуль із класом MainWindow, який реалізує головне вікно програми.

Головний блок програми - забезпечує виконання файлу як програми, а не імпортованого модуля.

Ініціалізація GUI:

- QApplication: створює та управляє подіями інтерфейсу;
- sys.argv: передає аргументи командного рядка.

Створення головного вікна:

- ініціалізується екземпляр класу MainWindow, де налаштовуються елементи інтерфейсу та логіка.

Запуск циклу обробки подій:

- app.exec_(): запускає події PyQt;
- sys.exit(): коректно завершує програму.

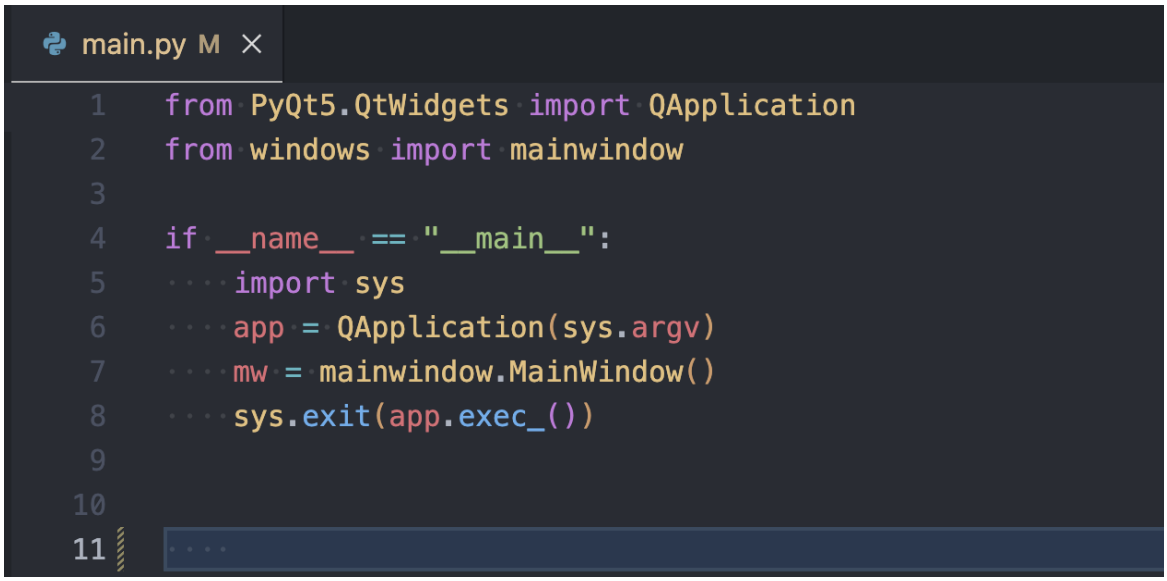

Рисунок 3.10 – Файл запуску програми “My_Antivirus”

Важливо:

- **відокремлення точки входу**: уся логіка графічного інтерфейсу знаходиться у файлі **mainwindow.py**, а файл **main.py** лише викликає його. Це дозволяє легко розширювати функціонал програми без зміни головного файлу.
- **масштабованість**: якщо в майбутньому потрібно додати інші модулі (наприклад, CLI-версію програми), це можна зробити, залишивши структуру незмінною.
- **зрозумілість коду**: мінімальний обсяг коду у файлі точки входу полегшує розуміння структури програми.

Графічний інтерфейс програми реалізовано у файлі **mainwindow.py** з використанням бібліотеки **PyQt5**. Розглянемо детально, як організовано роботу графічного інтерфейсу:

1. Основна структура графічного інтерфейсу.

Клас **MainWindow** - головний клас, який відповідає за створення основного вікна програми, його компонування, стилі та функціонал

2. Ключові компоненти інтерфейсу:

- **кнопки управління (QPushButton)**:

– використовуються для запуску основних дій (сканування файлів, відкриття карантину, налаштувань тощо);

– кнопки додаються в вертикальне компонування (QVBoxLayout) для зручного розташування.

- **текстове поле для результатів (QTextEdit)** - відображає результати сканування, повідомлення про помилки або іншу інформацію;
- **прогрес-бар (QProgressBar)** - показує прогрес сканування файлів чи папок;
- **таблиця карантину (QListWidget)** - містить список файлів, які були переміщені до карантину;
- **модальне вікно налаштувань (SettingsDialog)** - дає можливість вибору мови та теми оформлення.

2.1 Ініціалізація інтерфейсу.

Клас MainWindow розширює базовий клас PyQt5.QWidget.

```
207    def __init__(self):
208        super().__init__()
209        self.language = "English"
210        self.theme = "Light"
211        self.list_widget = QListWidget()
212        self.result_box = QTextEdit()
213        self.result_box.setReadOnly(True)
214        self.set_window_properties()
215        self.initUI()
216        self.connect_signals()
217        self.show()
```

Рисунок 3.11 – Ініціалізація інтерфейсу

- self.language і self.theme: визначають поточну мову та тему інтерфейсу;
- set_window_properties: задає загальні властивості вікна (розмір, заголовок);
- initUI: створює компоненти та додає їх до макету;
- connect_signals: прив'язує дії до кнопок.

2.2 Кнопки управління.

```
248         self.b_scan_file = QPushButton(self.translate("Scan File"))
249         self.b_scan_folder = QPushButton(self.translate("Scan Folder"))
250         self.b_quarantine = QPushButton(self.translate("Quarantine"))
251         self.b_exit = QPushButton(self.translate("Exit"))
252         self.b_settings = QPushButton(self.translate("Settings"), self)
253         self.b_delete = QPushButton(self.translate("Delete File"))
254         self.b_restore = QPushButton(self.translate("Restore File"))
255         self.b_scan_url = QPushButton(self.translate('Scan URL'))
```

Рисунок 3.12 – Кнопки програми

- всі кнопки оформлюються за допомогою CSS через метод **setStyleSheet**.

```
258         for button in [
259             self.b_scan_file, self.b_scan_folder, self.b_quarantine,
260             self.b_exit, self.b_settings, self.b_delete, self.b_restore, self.b_scan_url
261         ]:
262             button.setStyleSheet(button_style)
```

Рисунок 3.13 – Оформлення кнопок

```
225     def initUI(self):
226         button_style = """
227     QPushButton {
228             background: #7079f0;
229             color: white;
230             min-width: 150px;
231             font-size: 16px;
232             font-weight: 500;
233             border-radius: 0.5em;
234             border: none;
235             height: 2.5em;
236     }
237
238     QPushButton:hover {
239             background: #5b65f5;
240     }
241
242     QPushButton:pressed {
243         background: #404df7;
244     }
245     """
```

Рисунок 3.14 – Стилізація кнопок

- додаються в **QVBoxLayout**, щоб компоненти були вирівняні по вертикалі;
- Пов'язуються з діями за допомогою сигналів:

```
401     def connect_signals(self):
402         self.b_scan_file.clicked.connect(self.scan_file)
403         self.b_scan_folder.clicked.connect(self.scan_folder)
404         self.b_quarantine.clicked.connect(self.show_quarantine)
405         self.b_exit.clicked.connect(QApplication.quit)
406         self.b_settings.clicked.connect(self.open_settings)
407         self.b_delete.clicked.connect(self.delete_selected_file)
408         self.b_restore.clicked.connect(self.restore_selected_file)
409         self.b_scan_url.clicked.connect(self.scan_url)
```

Рисунок 3.15 – Додавання події на клік на кнопки

2.3 Текстове поле для результатів.

```
212         self.result_box = QTextEdit()
213         self.result_box.setReadOnly(True)
```

Рисунок 3.16 – Ініціалізація текстового поля для результатів сканування

- використовується для відображення результатів сканування;
- стилі змінюються залежно від вибраної теми.

2.4 Прогрес-бар.

```
366     def create_progress_bar(self):
367         bar = QProgressBar()
368         bar.setValue(0)
369         bar.setStyleSheet('''
370             QProgressBar { border: 2px solid #7079f0; border-radius: 0.5em;
                text-align: center; color: black; background: #fff; }
371             QProgressBar::chunk { background: #7079f0; border-radius: 0.2em;}
372         ''')
373         return bar
```

Рисунок 3.17 – Індикатор виконання сканування

- відображає прогрес під час сканування;
- прогрес оновлюється методом:

```
822         self.progress_bar.setValue(int((self.processed_files / total_files) * 100))
```

Рисунок 3.18 – Оновлення індикатору сканування

2.5 Налаштування теми та мови.

Клас SettingsDialog дозволяє змінювати:

- тему: темну або світлу;
- мову: англійську або українську.

```
39        self.language_label = QLabel(self.translate("Language:"))
40        self.language_combo = QComboBox()
41        self.language_combo.addItems(["English", "Українська"])
42        self.language_combo.setCurrentText(self.language)
43
44        self.theme_label = QLabel(self.translate("Theme:"))
45        self.theme_combo = QComboBox()
46        self.theme_combo.addItems([self.translate("Light"), self.translate("Dark")])
47
48        self.theme_combo.setCurrentText(self.theme)
```

Рисунок 3.19 – Вікно налаштувань програми

Залежно від вибраної теми, стилі компонентів змінюються:

```
384        if self.theme in ["Dark", "Темна"]:
385            self.list_widget.setStyleSheet('''
386                QListWidget { background: #222; border: 1px solid #555; font-size:
                   14px; color: white; }
387                QListWidget::item { padding: 5px; }
388                QListWidget::item:selected { background: #5055f0; color: white; }
389            ''')
390        else:
391            self.list_widget.setStyleSheet('''
392                QListWidget { background: white; border: 1px solid #ccc; font-size:
                   14px; color: black; }
393                QListWidget::item { padding: 5px; }
394                QListWidget::item:selected { background: #7079f0; color: white; }
395            ''')
```

Рисунок 3.20 – Стилізація компонентів в залежності від теми

3. Розташування елементів:

- зліва - панель з кнопками;
- справа - текстове поле, прогрес-бар, карантин.

```
273        self.main_layout = QHBoxLayout()
274        self.main_layout.addLayout(buttons_layout, stretch=1)
275        self.right_layout = self.create_right_layout()
276        self.main_layout.addLayout(self.right_layout, stretch=2)
277        self.setLayout(self.main_layout)
```

Рисунок 3.21 – Компонування програми

4. Переклади (translate).

```
279     def translate(self, text):
280         translations = {
281             "Scan File": "Сканувати файл",
282             "Scan Folder": "Сканувати папку",
283             "Quarantine": "Карантин",
284             "Exit": "Вихід",
285             "Settings": "Налаштування",
286             "Delete File": "Видалити файл",
287             "Restore File": "Відновити файл",
288             "Scan URL": "Сканувати URL",
289             "Enter URL for scanning:": "Введіть URL для сканування:",
290             "Scan Results for file": "Результат сканування файлу:",
291             "Scan Results for URL:": "Результати сканування URL-адреси:",
292             "Scan Error": "Помилка сканування",
293             ...
294         }
295         if self.language == "Українська":
296             return translations.get(text, text)
297         return text
```

Рисунок 3.22 – Функція перекладу мови програми

5. Підтримка асинхронного сканування.

Клас **FileScanThread** забезпечує виконання сканування файлів у фоновому режимі. Це важливо, щоб користувач міг взаємодіяти з інтерфейсом навіть під час тривалих операцій.

```
13  class FileScanThread(QThread):
14      progress = pyqtSignal(str, dict)
15      finished = pyqtSignal()
16
17      def __init__(self, file_path):
18          super().__init__()
19          self.file_path = file_path
20
21      def run(self):
22          result = virustotal.upload_file(self.file_path)
23          self.progress.emit(self.file_path, result)
24          self.finished.emit()
```

Рисунок 3.23 – Асинхронне сканування

6. Робота з карантином.

Карантин — це ключова функція антивірусу, яка містить інформацію про файли, які були переміщені в ізольоване місце.

Збереження карантину - лог зберігається у файлі **quarantine_log.json:**

```
981     def move_to_quarantine(self, file_path):
982         quarantine_path = os.path.join(self.QUARANTINE_FOLDER, os.path.basename(file_path))
983         os.makedirs(self.QUARANTINE_FOLDER, exist_ok=True)
984         shutil.move(file_path, quarantine_path)
985 
986         file_info = {
987             "file_name": os.path.basename(file_path),
988             "quarantine_path": quarantine_path,
989             "original_path": file_path,  # Сохраняем оригинальный путь
990             "date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
991             "status": "quarantined"
992         }
993 
994         if os.path.exists(self.QUARANTINE_LOG):
995             with open(self.QUARANTINE_LOG, "r") as f:
996                 log_data = json.load(f)
997         else:
998             log_data = []
999 
1000        log_data.append(file_info)
1001        with open(self.QUARANTINE_LOG, "w") as f:
1002            json.dump(log_data, f, indent=4)
```

Рисунок 3.24 – Ініціалізація карантину

Функції карантину:

– видалення файлів;

– відновлення файлів.

```
1004    def delete_selected_file(self):
1005        selected_item = self.quarantine_table.currentItem()
1006        if selected_item:  # Check if an item is selected
1007            file_entry = selected_item.data(Qt.UserRole)
1008            file_path = file_entry["quarantine_path"]
1009            if os.path.exists(file_path):
1010                os.remove(file_path)
1011                self.result_box.append(f"{self.translate('File deleted')}: {file_path}")
1012                self.remove_file_from_log(file_entry)
1013                self.load_quarantine()
1014            else:
1015                self.show_file_not_found_message(file_path)
1016        else:
1017            self.show_no_file_selected_message()
```

Рисунок 3.25 – Функція видалення файлів

```
1019        def restore_selected_file(self):
1020            selected_item = self.quarantine_table.currentItem()
1021            if selected_item:  # Check if an item is selected
1022                file_entry = selected_item.data(Qt.UserRole)
1023                file_path = file_entry["quarantine_path"]
1024                original_path = file_entry.get("original_path")
1025                if os.path.exists(file_path):
1026                    os.makedirs(os.path.dirname(original_path), exist_ok=True)
1027                    shutil.move(file_path, original_path)
1028                    self.result_box.append(f"{self.translate('File restored to')}:
                        {original_path}")
1029                    self.remove_file_from_log(file_entry)
1030                    self.load_quarantine()
1031                else:
1032                    self.show_file_not_found_message(file_path)
1033            else:
1034                self.show_no_file_selected_message()
```

Рисунок 3.26 – Функція відновлення файлів

7. Перевірка URL:

```
478     def scan_url(self):
479         dialog = QInputDialog(self)
480         dialog.setWindowTitle(f"{self.translate('Scan URL')}")
481         dialog.setLabelText(f"{self.translate('Enter URL for scanning:')}")
482         dialog.setCancelButtonText(self.translate("Cancel"))
483         dialog.resize(400, 200)
```

Рисунок 3.27 – Функція перевірки URL-адрес

```
621     def display_url_scan_result(self, url, result):
622         self.result_box.clear()
623         self.result_box.append(self.translate("Scan Results for URL:") + f" {url}")
624         self.result_box.append("=" * 50)
625
626         if "error" in result:
627             self.result_box.append(f"Scan Error: {result['error']}")
628             return
629
630         analysis_data = result.get("data", {}).get("attributes", {})
631         malicious_count = analysis_data.get("stats", {}).get("malicious", 0)
632         reputation = self.translate(analysis_data.get("reputation", "Unknown"))
633         last_analysis_results = analysis_data.get("last_analysis_results", {})
634
635         self.result_box.append(f"{self.translate('Malicious Reports')}: {malicious_count}")
636         self.result_box.append(f"{self.translate('Reputation')}: {reputation}")
637         self.result_box.append(self.translate("Detailed Results:"))
638
639         for engine, details in last_analysis_results.items():
640             category = details.get("category", "unknown")
641             result = details.get("result", "clean")
642             self.result_box.append(f"{engine}: {category} ({result})")
643
644         if malicious_count > 0:
645             self.result_box.append(self.translate("❌ URL is flagged as malicious!"))
646         else:
647             self.result_box.append(self.translate("✅ URL appears safe."))
648
649         self.result_box.append("=" * 50)
650         self.result_box.append(f"{self.translate('Scanning complete')}")
```

Рисунок 3.28 – Функція виводу результатів про сканування url-адрес

8. Темна та світла тема інтерфейсу змінюються динамічно:

- темна тема**:**

```
    def get_theme_stylesheet(self):
        if self.theme in ["Dark", "Темна"]:
            self.list_widget.setStyleSheet('''
                QListWidget { background: #222; border: 1px solid #555; font-size: 14px; color: white; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #5055f0; color: white; }
            ''')
            self.result_box.setStyleSheet(
                'background: rgba(30, 30, 30, 1); border: 1px solid #666; border-radius: 0.5em; font-size: 14px; color: white;'
            )
            return """
            QWidget { background: #333; color: white; }
            QPushButton { background: #555; color: white; border: 1px solid #777; }
            QPushButton:hover { background: #666; }
            QPushButton:pressed { background: #444; }
            """
```

Рисунок 3.29 – Стилізація темної теми програми

- світла тема**:**

```
        else:
            self.result_box.setStyleSheet(
                'background: white; border: 1px solid #ccc; border-radius: 0.5em; font-size: 14px; color: black;'
            )
            self.list_widget.setStyleSheet('''
                QListWidget { background: white; border: 1px solid #ccc; font-size: 14px; color: black; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #7079f0; color: white; }
            ''')
            return """
            QWidget { background: #EEEFF0; color: black; }
            QPushButton { background: #ccc; color: black; border: 1px solid #aaa; }
            QPushButton:hover { background: #ddd; }
            QPushButton:pressed { background: #bbb; }
            """
```

Рисунок 3.30 – Стилізація світлої теми програми

9. Обробка помилокю.

Інтерфейс надає користувачу зворотний зв'язок у разі помилок:

- помилка під час сканування**:**

```
    def handle_scan_result(self, file_path, result):
        self.result_box.append(f"{self.translate('Scan Results for file')}: {file_path}")
        self.result_box.append("=" * 50)

        if "error" in result:
            error_message = result["error"]
            self.result_box.append(f"{self.translate('Scan Error')}: {error_message}")
```

Рисунок 3.31 - Помилка під час сканування

- надто багато запитів:

```
            if "429" in error_message or "Too Many Requests" in error_message:
                self.result_box.append(f"{self.translate('⛔ Error 429: Too many requests.')}")
                retry = self.show_retry_dialog(file_path)
                if retry:
                    self.retry_scan(file_path)
                else:
                    self.result_box.append(f"{self.translate('Skipped')}: {file_path}")
            else:
                self.result_box.append(f"{self.translate('An error occurred. Skipping file.')}")
```

Рисунок 3.32 – Помилка (надто багато запитів) під час сканування

- не обрано жодного файлу:

```
    def show_no_file_selected_message(self):
        msg = QMessageBox(self)
        msg.setIcon(QMessageBox.Warning)
        msg.setWindowTitle(self.translate("No File Selected"))
        msg.setText(self.translate("Please select a file to delete or restore."))
        msg.setStandardButtons(QMessageBox.Ok)
        ok_button = msg.button(QMessageBox.Ok)
        if ok_button:
            ok_button.setStyleSheet("""
            QPushButton {
                background-color: #7079f0;
                width: 100px;
                color: white;
                font-size: 14px;
                font-weight: bold;
                border-radius: 8px;
                padding: 6px 12px;
                border: none;
            }
            QPushButton:hover {
                background-color: #5b65f5;
            }
            QPushButton:pressed {
                background-color: #404df7;
            }
        """)
        msg.exec_()
```

Рисунок 3.33 – Помилка (не обрано жодного файлу) для дії

10. Діалогові вікна

Діалогові вікна використовуються для підтвердження дій користувача:

- **діалог для заражених файлів** – в разі якщо файл вірусний, користувач обирає наступну дію для цього файлу: видалення, перемістити до карантину або пропустити файл.

```
835     def show_infected_file_dialog(self, file_path):
836         msg_box = QMessageBox(self)
837         msg_box.setIcon(QMessageBox.Warning)
838         msg_box.setWindowTitle(self.translate("Threat Detected"))
839         msg_box.setText(f"{self.translate('The file')} {file_path} {self.translate('is infected.')}")
840         msg_box.setInformativeText(self.translate("Select an action:"))
841
842         quarantine_button = msg_box.addButton(self.translate("Quarantine"), QMessageBox.AcceptRole)
843         delete_button = msg_box.addButton(self.translate("Delete"), QMessageBox.DestructiveRole)
844         skip_button = msg_box.addButton(self.translate("Skip"), QMessageBox.RejectRole)
```

Рисунок 3.34 – Діалогове вікно для вибору дії вірусному файлу

- **діалог для налаштувань програми:**

```
26  class SettingsDialog(QDialog):
27      def __init__(self, current_language, current_theme, parent=None):
28          super().__init__(parent)
29          self.language = current_language
30          self.theme = current_theme
31          self.setWindowTitle(self.translate("Settings"))
32          self.resize(400, 200)
33
34          self.apply_theme_styles()
35
36          self.language_label = QLabel(self.translate("Language:"))
37          self.language_combo = QComboBox()
38          self.language_combo.addItems(["English", "Українська"])
39          self.language_combo.setCurrentText(self.language)
40
41          self.theme_label = QLabel(self.translate("Theme:"))
42          self.theme_combo = QComboBox()
43          self.theme_combo.addItems([self.translate("Light"), self.translate("Dark")])
44
45          self.theme_combo.setCurrentText(self.theme)
```

Рисунок 3.35 – Діалогове вікно налаштувань програми

- **діалог для помилки в карантині:**

```
983     def show_no_file_selected_message(self):
984         msg = QMessageBox(self)
985         msg.setIcon(QMessageBox.Warning)
986         msg.setWindowTitle(self.translate("No File Selected"))
987         msg.setText(self.translate("Please select a file to delete or restore."))
988         msg.setStandardButtons(QMessageBox.Ok)
989         ok_button = msg.button(QMessageBox.Ok)
```

Рисунок 3.36 – Діалогове вікно для помилки в карантині

ВИСНОВОК

У даній кваліфікаційній роботі магістра було:

• проведено комплексне дослідження у сфері захисту комп'ютерних інформаційних систем, зокрема методів виявлення та запобігання кіберзагрозам;

- проаналізовані сучасні підходи до розробки антивірусного програмного забезпечення, включаючи методи статичного та динамічного аналізу;
- визначено основні вимоги до антивірусних програм у контексті сучасних загроз, зокрема їхню здатність до інтеграції з існуючими платформами, продуктивність і зручність користувацького інтерфейсу.

На основі проведеного теоретичного аналізу було розроблено програмний продукт «**My_Antivirus**», що реалізує сучасні підходи до антивірусного захисту.

1. **Основні функції програми**:

- **сканування файлів та папок** - використання інтеграції з платформою VirusTotal для виявлення шкідливих об'єктів;
- **аналіз URL-адрес** - оцінювання URL на предмет їхньої безпечності за допомогою API;
- **карантин** - ізоляція небезпечних файлів із можливістю їх видалення або відновлення;
- **динамічне оновлення інтерфейсу** - асинхронне виконання операцій для зручності користувача;
- **підтримка багатомовності та світлого/темного режиму**.

2. Також було проведено **випробування програми** за такими вимогами:

- перевірка **точності виявлення загроз** на тестовому наборі файлів;
- **аналіз продуктивності** при обробці великого обсягу даних;
- **оцінка зручності** користувацького інтерфейсу.

Таким чином, виконана робота має значний науковий і практичний потенціал. Програмний продукт **"My_Antivirus"** демонструє ефективність сучасних підходів до антивірусного захисту, поєднуючи інтеграцію з популярними платформами (VirusTotal), аналіз URL-адрес, ізоляцію загроз у карантині та інтуїтивно зрозумілий багатомовний інтерфейс.

Проведені випробування підтвердили здатність програми виявляти загрози, обробляти великі обсяги даних без втрати продуктивності та забезпечувати зручність використання. Цей прототип може слугувати основою для створення повноцінного антивірусного рішення з розширеним функціоналом, що сприятиме розвитку сфери інформаційної безпеки.

Тема даної кваліфікаційної роботи була апробована на **76-й науковій конференції (**14-23 травня 2024року), де у ході обговорення були висвітлені ключові переваги запропонованих рішень, а також надані рекомендації для їх удосконалення. На основі отриманих висновків була виконана дана кваліфікаційна робота, яка доповнює попереднє дослідження новими практичними рішеннями. Це дозволило підтвердити актуальність теми, перевірити її практичну значимість і врахувати зауваження для подальшого розвитку проєкту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Беляков К. Інформація організаційно-правової сфери /К.Беляков // Право України. —2004. —№ 6. — С. 88-92.

2. Кормич Б.А. Інформаційна безпека: організаційно-правові основи: Навч. посібник. - К.: Кондор, 2004. - 384 с.

3. Харченко В. С. Інформаційна безпека. Глосарій. — К.: КНТ, 2005.

4. Закон Про комп'ютерну безпеку 1987 року. URL: https://www.ssa.gov/OP_Home/comp2/F100-235.html

5. Засоби захисту інформації. Системи розмежування доступу. URL: https://stud.com.ua/94403/informatika/zasobi_zahistu_informatsiyi.

6. Засоби та методи захисту інформації. URL: https://buklib.net/books/28625/.

7. Захист інформації. Види захисту інформації. URL: https://uk.wikipedia.org/wiki/Захист_інформації

8. Заходи щодо захисту від вірусів. Антивірусні програми. URL: https://stud.com.ua/94406/informatika/zahodi_schodo_zahistu_virusiv .

9. Класифікація комп'ютерних вірусів. Загальні властивості та поведінка вірусів. URL: https://studfile.net/preview/9228869/page:7/.

10. Комп'ютерний вірус. Історія створення комп'ютерних вірусів. URL: https://uk.wikipedia.org/wiki/Комп%27ютерний_вірус#Історія (дата звернення: 09.04.2023).

11. Основні види вірусних програм. URL: https://zillya.ua/index.php?q=osnovni-vidi-virusnikh-program .

12. Основні джерела вірусів. Класифікація вірусів. URL: https://studfile.net/preview/7357935/page:7/.

13. Атаки типу Man-In-The-Middle: що треба знати кожному. Домени – перевірка та реєстрація доменів в Україні | Imena.ua. URL: https://uk.wikipedia.org/wiki/Атака_«людина_посередині»

14. Бібліотеки Python: потужні рішення у розробці ПЗ. URL: https://www.python.org/

15. Введення в тестування програмного забезпечення | Q & A. Q & A - Навчальний ресурс з тестування програмного забезпечення. URL: https://qalearning.com.ua/category/theory/lectures/material/

16. Загрози при роботі в інтернеті і їх уникнення. Освітній проект «На Урок» для вчителів. URL: https://naurok.com.ua/test/informaciyna-bezpeka-zagrozi-pri-roboti-v-interneti-i-h-uniknennya-573482.html

17. Люта, Майя В'ячеславівна, Інна Олександрівна Розломій та Катерина Володимирівна Новикова. «Аналіз методів тестування програмного забезпечення». Thesis, Міжнародний центр наукових досліджень, 2017. URL: https://er.knutd.edu.ua/handle/123456789/8421.

18. Термінологічний довідник з питань технічного захисту інформації / Коженевський С.Р., Кузнецов Г.В., Хорошко В. О., Чирков Д.В. / За ред. пр оф. В. О. Хорошка. – К.: ДУ ІКТ, 2007. – 365 с.

19. Що таке інформаційна безпека (InfoSec)? | Захисний комплекс Microsoft. Microsoft. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-information-security-infosec

20. Microsoft. Language support in visual studio code. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com/docs/languages/overview

21. Virus Share [Electronic resource]. – Access mode: https://virusshare.com/

V

i 23. Zillya! Антивірус — український антивірус https://zillya.ua/

r 24. Avast | Free Antivirus, VPN https://www.avast.ua/ru-ua/index#mac

u 25. Windows Defender https://support.microsoft.com/fr-fr/topic/installation-de-microsoft-defender-ba7d17d7-464b-444d-bb47-990b7278369d

T 26. Bitdefender Internet Security https://www.bitdefender.com/en-us/consumer/internet-security

t 27. Avira Internet Security https://www.avira.com

a

l

ДОДАТОК А
ЛІСТИНГ ПРОГРАМИ

Файл запуску програми main.py:

```
from PyQt5.QtWidgets import QApplication
from windows import mainwindow

if __name__ == "__main__":
    import sys
    app = QApplication(sys.argv)
    mw = mainwindow.MainWindow()
    sys.exit(app.exec_())
```

Інтеграція з Virustotal у файлі virustotal.py:

```
import requests

API_KEY = '53fc8fcb34397a326376729f594ce29fae66a137ba312f6bf4854ec385dcd67b'
UPLOAD_URL = "https://www.virustotal.com/api/v3/files"
FILE_INFO_URL = "https://www.virustotal.com/api/v3/files"
BASE_URL = "https://www.virustotal.com/api/v3"

def upload_file(path):
    try:
        with open(path, "rb") as f:
            files = {"file": (path, f)}
            headers = {"x-apikey": API_KEY}
            response = requests.post(UPLOAD_URL, headers=headers, files=files)

        if response.status_code == 200:
            file_id = response.json().get("data", {}).get("id")
            if not file_id:
                return {"error": "File ID not found in upload response."}

            result_url = f"https://www.virustotal.com/api/v3/analyses/{file_id}"
            result_response = requests.get(result_url, headers={"x-apikey": API_KEY})

            if result_response.status_code == 200:
                data = result_response.json().get("data", {}).get("attributes", {})
                engine_results = {
                    "malicious": [],
                    "harmless": [],
                    "suspicious": [],
```

```
                "undetected": []
            }

            last_analysis_results = data.get("results", {})
            for engine, result in last_analysis_results.items():
                category = result.get("category")
                engine_info = {
                    "engine": engine,
                    "result": result.get("result", "N/A"),
                    "engine_version": result.get("engine_version", "N/A"),
                    "update": result.get("update", "N/A"),
                }
                if category in engine_results:
                    engine_results[category].append(engine_info)

            return {
                "malicious_count": len(engine_results["malicious"]),
                "harmless_count": len(engine_results["harmless"]),
                "suspicious_count": len(engine_results["suspicious"]),
                "undetected_count": len(engine_results["undetected"]),
                "engine_results": engine_results
            }
        else:
            return {"error": f"Error fetching analysis results: {result_response.status_code}"}
    else:
        return {"error": f"Error uploading file: {response.status_code}"}
except Exception as e:
    return {"error": str(e)}

def parse_analysis_results(results):
    if not results:
        return {"error": "No analysis results available."}

    engine_results = {
        "malicious": [],
        "clean": [],
        "undetected": []
    }

    for engine, result in results.items():
        category = result.get("category")
        engine_info = {
            "engine": engine,
            "result": result.get("result", "N/A"),
            "version": result.get("engine_version", "N/A"),
            "update": result.get("engine_update", "N/A")
        }
        if category == "malicious":
```

```
            engine_results["malicious"].append(engine_info)
        elif category == "clean":
            engine_results["clean"].append(engine_info)
        else:
            engine_results["undetected"].append(engine_info)

    return {
        "malicious_count": len(engine_results["malicious"]),
        "clean_count": len(engine_results["clean"]),
        "undetected_count": len(engine_results["undetected"]),
        "engine_results": engine_results
    }

def scan_url(url):
    headers = {"x-apikey": API_KEY}
    submit_url = f"{BASE_URL}/urls"

    response = requests.post(submit_url, headers=headers, data={"url": url})
    if response.status_code != 200:
        return {"error": f"Error submitting URL: {response.status_code} {response.text}"}

    analysis_id = response.json().get("data", {}).get("id")
    if not analysis_id:
        return {"error": "Failed to retrieve analysis ID."}

    result_url = f"{BASE_URL}/analyses/{analysis_id}"
    result_response = requests.get(result_url, headers=headers)
    if result_response.status_code != 200:
        return {"error": f"Error fetching analysis results: {result_response.status_code}
{result_response.text}"}

    return result_response.json()
```

Основний файл програми mainwindow.py:

```
from PyQt5.QtWidgets import (
    QWidget, QPushButton, QFileDialog, QApplication, QProgressBar, QTextEdit, QHBoxLayout,
QVBoxLayout, QListWidgetItem, QMessageBox, QListWidget, QInputDialog, QDialog, QComboBox,
QLabel, QDialogButtonBox, QLineEdit
)
from PyQt5.QtCore import Qt
import os
import json
import shutil
from datetime import datetime
from helpers import virustotal
from PyQt5.QtCore import QThread, pyqtSignal
import time
```

```
class FileScanThread(QThread):
    progress = pyqtSignal(str, dict)
    finished = pyqtSignal()

    def __init__(self, file_path):
        super().__init__()
        self.file_path = file_path

    def run(self):
        result = virustotal.upload_file(self.file_path)
        self.progress.emit(self.file_path, result)
        self.finished.emit()

class SettingsDialog(QDialog):
    def __init__(self, current_language, current_theme, parent=None):
        super().__init__(parent)
        self.language = current_language
        self.theme = current_theme
        self.setWindowTitle(self.translate("Settings"))
        self.resize(400, 200)

        self.apply_theme_styles()

        self.language_label = QLabel(self.translate("Language:"))
        self.language_combo = QComboBox()
        self.language_combo.addItems(["English", "Українська"])
        self.language_combo.setCurrentText(self.language)

        self.theme_label = QLabel(self.translate("Theme:"))
        self.theme_combo = QComboBox()
        self.theme_combo.addItems([self.translate("Light"), self.translate("Dark")])

        self.theme_combo.setCurrentText(self.theme)

        layout = QVBoxLayout()
        layout.addWidget(self.language_label)
        layout.addWidget(self.language_combo)
        layout.addWidget(self.theme_label)
        layout.addWidget(self.theme_combo)

        self.ok_button = QPushButton(self.translate("Ok"))
        self.cancel_button = QPushButton(self.translate("Cancel"))
        self.ok_button.clicked.connect(self.apply_settings)
        self.cancel_button.clicked.connect(self.reject)

        button_layout = QHBoxLayout()
        button_layout.addWidget(self.ok_button)
```

```
    button_layout.addWidget(self.cancel_button)

    layout.addLayout(button_layout)
    self.setLayout(layout)

def apply_theme_styles(self):
    if self.theme in ["Dark", "Темна"]:
        self.setStyleSheet("""
            QDialog {
                background-color: #333333;
                border: 1px solid #333333;
                border-radius: 8px;
            }
            QLabel {
                color: white;
                font-size: 14px;
            }
            QComboBox {
                background-color: #1E1E1E;
                color: white;
                border: 1px solid #d3d3d3;
                border-radius: 4px;
                padding: 5px;
            }
            QComboBox QAbstractItemView {
                background-color: #444444;
                color: white;
                selection-background-color: #5b65f5;
            }
            QPushButton {
                background: #7079f0;
                color: white;
                min-width: 100px;
                font-size: 14px;
                font-weight: 500;
                border-radius: 0.5em;
                border: none;
                height: 1.5em;
            }

            QPushButton:hover {
                background: #5b65f5;
            }

            QPushButton:pressed {
                background: #404df7;
            }
        """)
```

```
        else:
            self.setStyleSheet("""
                QDialog {
                    background-color: #EEEFF0;
                    border: 1px solid #d3d3d3;
                    border-radius: 8px;
                }
                QLabel {
                    color: #333;
                    font-size: 14px;
                }
                QComboBox {
                    background-color: #ffffff;
                    color: #000;
                    border: 1px solid #d3d3d3;
                    border-radius: 4px;
                    padding: 5px;
                }
                QComboBox QAbstractItemView {
                    background-color: #ffffff;
                    color: black;
                    selection-background-color: #7079f0;
                }
                QPushButton {
                    background: #7079f0;
                    color: white;
                    min-width: 100px;
                    font-size: 14px;
                    font-weight: 500;
                    border-radius: 0.5em;
                    border: none;
                    height: 1.5em;
                }

                QPushButton:hover {
                    background: #5b65f5;
                }

                QPushButton:pressed {
                    background: #404df7;
                }
            """)

    def apply_settings(self):
        self.language = self.language_combo.currentText()
        self.theme = self.theme_combo.currentText()
        self.accept()
```

```
    def get_settings(self):
        return {
            "language": self.language_combo.currentText(),
            "theme": self.theme_combo.currentText()
        }

    def translate(self, text):
        translations = {
            "Settings": "Налаштування",
            "Language:": "Мова:",
            "Theme:": "Тема:",
            "Ok": "Ok",
            "Cancel": "Відміна",
            "Light": "Світла",
            "Dark": "Темна",
        }
        if self.language == "Українська":
            return translations.get(text, text)
        return text

class QuarantineWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.theme = "Light"
        self.list_widget = self.create_quarantine_table(self.theme)
        layout = QVBoxLayout()
        layout.addWidget(self.list_widget)
        self.setLayout(layout)

    def update_theme(self, theme):
        self.theme = theme
        self.list_widget.setParent(None)
        self.list_widget = self.create_quarantine_table(self.theme)
        self.layout().addWidget(self.list_widget)

class MainWindow(QWidget):
    QUARANTINE_FOLDER = "quarantine"
    QUARANTINE_LOG = "quarantine_log.json"

    def __init__(self):
        super().__init__()
        self.language = "English"
        self.theme = "Light"
        self.list_widget = QListWidget()
        self.result_box = QTextEdit()
        self.result_box.setReadOnly(True)
        self.set_window_properties()
        self.initUI()
```

```
    self.connect_signals()
    self.show()

def set_window_properties(self):
    self.setWindowTitle('My_Antivirus')
    self.resize(800, 600)
    self.setStyleSheet(self.get_theme_stylesheet())

def initUI(self):
    button_style = """
QPushButton {
        background: #7079f0;
        color: white;
        min-width: 150px;
        font-size: 16px;
        font-weight: 500;
        border-radius: 0.5em;
        border: none;
        height: 2.5em;
}
QPushButton:hover {
        background: #5b65f5;
}
QPushButton:pressed {
    background: #404df7;
}
"""

    self.b_scan_file = QPushButton(self.translate("Scan File"))
    self.b_scan_folder = QPushButton(self.translate("Scan Folder"))
    self.b_quarantine = QPushButton(self.translate("Quarantine"))
    self.b_exit = QPushButton(self.translate("Exit"))
    self.b_settings = QPushButton(self.translate("Settings"), self)
    self.b_delete = QPushButton(self.translate("Delete File"))
    self.b_restore = QPushButton(self.translate("Restore File"))
    self.b_scan_url = QPushButton(self.translate('Scan URL'))

    for button in [
        self.b_scan_file, self.b_scan_folder, self.b_quarantine,
        self.b_exit, self.b_settings, self.b_delete, self.b_restore, self.b_scan_url
    ]:
        button.setStyleSheet(button_style)

    buttons_layout = QVBoxLayout()
    buttons_layout.addWidget(self.b_scan_file)
    buttons_layout.addWidget(self.b_scan_folder)
    buttons_layout.addWidget(self.b_quarantine)
    buttons_layout.addWidget(self.b_scan_url)
```

```
        buttons_layout.addWidget(self.b_settings)
        buttons_layout.addWidget(self.b_exit)

        self.main_layout = QHBoxLayout()
        self.main_layout.addLayout(buttons_layout, stretch=1)
        self.right_layout = self.create_right_layout()
        self.main_layout.addLayout(self.right_layout, stretch=2)
        self.setLayout(self.main_layout)

    def translate(self, text):
        translations = {
            "Scan File": "Сканувати файл",
            "Scan Folder": "Сканувати папку",
            "Quarantine": "Карантин",
            "Exit": "Вихід",
            "Settings": "Налаштування",
            "Delete File": "Видалити файл",
            "Restore File": "Відновити файл",
            "Scan URL": "Сканувати URL",
            "Enter URL for scanning:": "Введіть URL для сканування:",
            "Scan Results for file": "Результат сканування файлу:",
            "Scan Results for URL:": "Результати сканування URL-адреси:",
            "Scan Error": "Помилка сканування",
            "☐ Error 429: Too many requests.": "☐ Помилка 429: Забагато запитів.",
            "Skipped": "☐ Файл пропущено",
            "☐ File deleted": "☐ Файл видалено",
            "☐ File moved to quarantine": "☐ Файл переміщено до карантину",
            "An error occurred. Skipping file.": "Сталася помилка. Пропускаємо файл.",
            "❌Malicious file detected": "❌Виявлено шкідливий файл",
            "☐ Number of security providers": "☐ Кількість антивірусів, які виявили шкідливий файл:",
            "✅File is safe.": "✅Файл безпечний.",
            "General result: All files are safe.": "Загальний результат: Всі файли безпечні",
            "General result: Found": "Загальний результат: Знайдено",
            " infected file(s).": " заражений(их) файл(ів).",
            "Scanning complete": "Сканування завершено",
            "Quarantine is empty": "Карантин пустий",
            "Cancel": "Відміна",
            "Malicious Reports": "Зловмисні звіти",
            "Reputation": "Репутація",
            "reputation": "Репутація",
            "Unknown": "Невідома",
            "Detailed Results:": "Результат сканування:",
            "❌URL is flagged as malicious!": "❌URL позначений як зловмисний",
            "✅URL appears safe.": "✅URL виглядає безпечним",
            "Quarantine": "В карантин",
            "Delete": "Видалити",
            "Skip": "Пропустити",
            "Retry": "Повторити",
```

```
            "Select an action:": "Оберіть дію:",
            "Threat Detected": "Виявлено загрозу",
            "The file": "Файл",
            "is infected.": "є зловмисним",
            "Too Many Requests": "Забагато запитів",
            "Would you like to try scanning it again?": "Бажаєте спробувати відсканувати його ще раз?",
            "could not be scanned due to too many requests.": "не вдалося відсканувати через збільшену кількість запитів.",
            "Failed to scan": "Не вдалося просканувати",
            "after": "після",
            "attempts. Skipping file.": "спроб. Пропускаємо файл.",
            "Scan Error:": "Помилка сканування:",
            "error": "помилка",
            "Total checks:": "Усього перевірок:",
            "Unknown action for": "Невідома дія для",
            "File deleted": "Файл видалено",
            "File restored to": "Файл відновлено до",
            "No File Selected": "Файл не вибрано",
            "Please select a file to delete or restore.": "Будь ласка, виберіть файл для видалення або відновлення.",
            "File Not Found": "Файл не знайдено",
            "was not found.": "не було знайдено.",
        }
        if self.language == "Українська":
            return translations.get(text, text)
        return text


    def create_right_layout(self):
        layout = QVBoxLayout()
        self.progress_bar = self.create_progress_bar()
        self.result_box = self.create_result_box()
        self.quarantine_table = self.create_quarantine_table()

        layout.addWidget(self.progress_bar)
        layout.addWidget(self.result_box)
        layout.addWidget(self.quarantine_table)

        self.b_delete.setStyleSheet(self.b_restore.styleSheet())
        self.b_restore.setStyleSheet(self.b_delete.styleSheet())
        self.b_delete.setVisible(False)
        self.b_restore.setVisible(False)

        layout.addWidget(self.b_delete)
        layout.addWidget(self.b_restore)
        return layout

    def create_progress_bar(self):
```

```
        bar = QProgressBar()
        bar.setValue(0)
        bar.setStyleSheet('''
            QProgressBar { border: 2px solid #7079f0; border-radius: 0.5em; text-align: center; color: black;
background: #fff; }
            QProgressBar::chunk { background: #7079f0; border-radius: 0.2em;}
        ''')
        return bar

    def create_result_box(self):
        box = QTextEdit()
        box.setReadOnly(True)
        box.setStyleSheet('background: white; border-radius: 0.5em; border: 1px solid #ccc; font-size:
14px;')
        return box


    def create_quarantine_table(self):
        if self.theme in ["Dark", "Темна"]:
            self.list_widget.setStyleSheet('''
                QListWidget { background: #222; border: 1px solid #555; font-size: 14px; color: white; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #5055f0; color: white; }
            ''')
        else:
            self.list_widget.setStyleSheet('''
                QListWidget { background: white; border: 1px solid #ccc; font-size: 14px; color: black; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #7079f0; color: white; }
            ''')

        self.list_widget.hide()
        return self.list_widget

    def connect_signals(self):
        self.b_scan_file.clicked.connect(self.scan_file)
        self.b_scan_folder.clicked.connect(self.scan_folder)
        self.b_quarantine.clicked.connect(self.show_quarantine)
        self.b_exit.clicked.connect(QApplication.quit)
        self.b_settings.clicked.connect(self.open_settings)
        self.b_delete.clicked.connect(self.delete_selected_file)
        self.b_restore.clicked.connect(self.restore_selected_file)
        self.b_scan_url.clicked.connect(self.scan_url)

    def open_settings(self):
        settings_dialog = SettingsDialog(self.language, self.theme, self)
        if settings_dialog.exec_() == QDialog.Accepted:
            settings = settings_dialog.get_settings()
```

```
            self.language = settings["language"]
            self.theme = settings["theme"]
            self.update_ui()

    def update_ui(self):
        self.setStyleSheet(self.get_theme_stylesheet())
        self.b_scan_file.setText(self.translate("Scan File"))
        self.b_scan_folder.setText(self.translate("Scan Folder"))
        self.b_quarantine.setText(self.translate("Quarantine"))
        self.b_exit.setText(self.translate("Exit"))
        self.b_settings.setText(self.translate("Settings"))
        self.b_delete.setText(self.translate("Delete File"))
        self.b_restore.setText(self.translate("Restore File"))
        self.b_scan_url.setText(self.translate("Scan URL"))

    def get_theme_stylesheet(self):
        if self.theme in ["Dark", "Темна"]:
            self.list_widget.setStyleSheet('''
                QListWidget { background: #222; border: 1px solid #555; font-size: 14px; color: white; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #5055f0; color: white; }
            ''')
            self.result_box.setStyleSheet(
                'background: rgba(30, 30, 30, 1); border: 1px solid #666; border-radius: 0.5em; font-size: 14px;
color: white;'
            )
            return """
            QWidget { background: #333; color: white; }
            QPushButton { background: #555; color: white; border: 1px solid #777; }
            QPushButton:hover { background: #666; }
            QPushButton:pressed { background: #444; }
            """
        else:
            self.result_box.setStyleSheet(
                'background: white; border: 1px solid #ccc; border-radius: 0.5em; font-size: 14px; color: black;'
            )
            self.list_widget.setStyleSheet('''
                QListWidget { background: white; border: 1px solid #ccc; font-size: 14px; color: black; }
                QListWidget::item { padding: 5px; }
                QListWidget::item:selected { background: #7079f0; color: white; }
            ''')
            return """
            QWidget { background: #EEEFF0; color: black; }
            QPushButton { background: #ccc; color: black; border: 1px solid #aaa; }
            QPushButton:hover { background: #ddd; }
            QPushButton:pressed { background: #bbb; }
            """
```

```
def scan_file(self):
    self.show_scan_view()
    file_path, _ = QFileDialog.getOpenFileName(self, "Select File")
    if file_path:
        self.scan_files([file_path])

def scan_folder(self):
    self.show_scan_view()
    directory = QFileDialog.getExistingDirectory(self, "Select Folder")
    if directory:
        files = [os.path.join(root, file) for root, _, files in os.walk(directory) for file in files]
        self.scan_files(files)

def scan_url(self):
    dialog = QInputDialog(self)
    dialog.setWindowTitle(f"{self.translate('Scan URL')}")
    dialog.setLabelText(f"{self.translate('Enter URL for scanning:')}")
    dialog.setCancelButtonText(self.translate("Cancel"))
    dialog.resize(400, 200)
    self.result_box.show()
    self.quarantine_table.hide()

    if self.theme in ["Dark", "Темна"]:
        dialog.setStyleSheet("""
        QInputDialog {
            background-color: #333333;
            border: 1px solid #333333;
            border-radius: 8px;
        }
        QLabel {
            color: white;
            font-size: 14px;
        }
        QPushButton {
            background: #7079f0;
            color: white;
            min-width: 100px;
            font-size: 14px;
            font-weight: 500;
            border-radius: 0.5em;
            border: none;
            height: 1.5em;
        }
        QPushButton:hover {
            background: #5b65f5;
        }
        QPushButton:pressed {
            background: #404df7;
```

```
    }
""")
else:
    dialog.setStyleSheet("""
    QInputDialog {
        background-color: #EEEFF0;
        border: 1px solid #d3d3d3;
        border-radius: 8px;
    }
    QLabel {
        color: #333;
        font-size: 14px;
    }
    QPushButton {
        background: #7079f0;
        color: white;
        min-width: 100px;
        font-size: 14px;
        font-weight: 500;
        border-radius: 0.5em;
        border: none;
        height: 1.5em;
    }
    QPushButton:hover {
        background: #5b65f5;
    }
    QPushButton:pressed {
        background: #404df7;
    }
""")
line_edit = dialog.findChild(QLineEdit)
if line_edit:
    if self.theme in ["Dark", "Темна"]:
        line_edit.setStyleSheet("""
        QLineEdit {
            background-color: #1E1E1E;
            color: white;
            border: 1px solid #d3d3d3;
            border-radius: 4px;
            padding: 5px;
        }
    """)
    else:
        line_edit.setStyleSheet("""
        QLineEdit {
            background-color: white;
            color: #000;
            border: 1px solid #d3d3d3;
```

```
            border-radius: 4px;
            padding: 5px;
        }
    """)
button_box = dialog.findChild(QDialogButtonBox)
if button_box:
    cancel_button = button_box.button(QDialogButtonBox.Cancel)
    ok_button = button_box.button(QDialogButtonBox.Ok)
    if cancel_button:
        cancel_button.setStyleSheet("""
        QPushButton {
            background: #7079f0;
            color: white;
            min-width: 100px;
            font-size: 14px;
            font-weight: 500;
            border-radius: 0.5em;
            border: none;
            height: 1.5em;
        }
        QPushButton:hover {
            background: #5b65f5;
        }
        QPushButton:pressed {
            background: #404df7;
        }
    """)
    if ok_button:
        ok_button.setStyleSheet("""
        QPushButton {
            background: #7079f0;
            color: white;
            min-width: 100px;
            font-size: 14px;
            font-weight: 500;
            border-radius: 0.5em;
            border: none;
            height: 1.5em;
        }
        QPushButton:hover {
            background: #5b65f5;
        }
        QPushButton:pressed {
            background: #404df7;
        }
    """)

if dialog.exec_() == QInputDialog.Accepted:
```

```
            url = dialog.textValue()
            if url:
                self.result_box.append(f"Scanning URL: {url}\nPlease wait...")
                QApplication.processEvents()
                result = virustotal.scan_url(url)
                self.display_url_scan_result(url, result)
                self.progress_bar.setValue(100)
    def display_url_scan_result(self, url, result):
        self.result_box.clear()
        self.result_box.append(self.translate("Scan Results for URL:") + f" {url}")
        self.result_box.append("=" * 50)

        if "error" in result:
            self.result_box.append(f"Scan Error: {result['error']}")
            return

        analysis_data = result.get("data", {}).get("attributes", {})
        malicious_count = analysis_data.get("stats", {}).get("malicious", 0)
        reputation = self.translate(analysis_data.get("reputation", "Unknown"))
        last_analysis_results = analysis_data.get("last_analysis_results", {})

        self.result_box.append(f"{self.translate('Malicious Reports')}: {malicious_count}")
        self.result_box.append(f"{self.translate('Reputation')}: {reputation}")
        self.result_box.append(self.translate("Detailed Results:"))

        for engine, details in last_analysis_results.items():
            category = details.get("category", "unknown")
            result = details.get("result", "clean")
            self.result_box.append(f"{engine}: {category} ({result})")

        if malicious_count > 0:
            self.result_box.append(self.translate("❌URL is flagged as malicious!"))
        else:
            self.result_box.append(self.translate("✅URL appears safe."))

        self.result_box.append("=" * 50)
        self.result_box.append(f"{self.translate('Scanning complete')}")
        self.result_box.ensureCursorVisible()

    def show_scan_view(self):
        self.result_box.show()
        self.quarantine_table.hide()
        self.b_delete.setVisible(False)
        self.b_restore.setVisible(False)

    def show_quarantine(self):
        self.result_box.hide()
        self.quarantine_table.show()
```

```
        self.load_quarantine()

    def load_quarantine(self):
        self.quarantine_table.clear()
        if not os.path.exists(self.QUARANTINE_LOG):
            self.quarantine_table.addItem(f"{self.translate('Quarantine is empty')}")
            return

        with open(self.QUARANTINE_LOG, "r") as f:
            log_data = json.load(f)

        quarantined_files = [entry for entry in log_data if entry["status"] == "quarantined" and
os.path.exists(entry["quarantine_path"])]
        for entry in log_data:
            if entry["status"] == "quarantined" and not os.path.exists(entry["quarantine_path"]):
                self.remove_file_from_log(entry)

        if not quarantined_files:
            self.quarantine_table.addItem(f"{self.translate('Quarantine is empty')}")
        else:
            for file in quarantined_files:
                item = QListWidgetItem(f"{file['file_name']} - {file['quarantine_path']} ({file['date']})")
                item.setData(Qt.UserRole, file)
                self.quarantine_table.addItem(item)

        self.b_delete.setVisible(True)
        self.b_restore.setVisible(True)

    def remove_file_from_log(self, file_entry):
        if not os.path.exists(self.QUARANTINE_LOG):
            return

        with open(self.QUARANTINE_LOG, "r") as f:
            log_data = json.load(f)

        log_data = [entry for entry in log_data if entry != file_entry]
        with open(self.QUARANTINE_LOG, "w") as f:
            json.dump(log_data, f, indent=4)

    def scan_files(self, files):
        self.result_box.clear()
        self.progress_bar.setValue(0)
        self.threads = []
        self.processed_files = 0
        self.infected_files = []

        for file_path in files:
            thread = FileScanThread(file_path)
```

```
            thread.progress.connect(self.handle_scan_result)
            thread.finished.connect(self.update_progress_bar)
            self.threads.append(thread)
            thread.start()

    def handle_scan_result(self, file_path, result):
        self.result_box.append(f"{self.translate('Scan Results for file')}: {file_path}")
        self.result_box.append("=" * 50)

        if "error" in result:
            error_message = result["error"]
            self.result_box.append(f"{self.translate('Scan Error')}: {error_message}")

            if "429" in error_message or "Too Many Requests" in error_message:
                self.result_box.append(f"{self.translate('☐ Error 429: Too many requests.')}")
                retry = self.show_retry_dialog(file_path)
                if retry:
                    self.retry_scan(file_path)
                else:
                    self.result_box.append(f"{self.translate('Skipped')}: {file_path}")
            else:
                self.result_box.append(f"{self.translate('An error occurred. Skipping file.')}")
        else:
            malicious_count = result.get('malicious_count', 0)
            if malicious_count > 0:
                self.result_box.append(f"{self.translate('✗Malicious file detected')}: {file_path}")
                self.result_box.append(f"{self.translate('☐ Number of security providers')}: {malicious_count}")
                self.infected_files.append(file_path)

                action = self.show_infected_file_dialog(file_path)
                self.handle_infected_file_action(action, file_path)
            else:
                self.result_box.append(f"{self.translate('✓File is safe.')}")

        self.result_box.append("=" * 50 + "\n")

    def show_retry_dialog(self, file_path):
        msg_box = QMessageBox(self)
        msg_box.setIcon(QMessageBox.Warning)
        msg_box.setWindowTitle(self.translate("Too Many Requests"))
        msg_box.setText(f"{self.translate('The file')} {file_path} {self.translate('could not be scanned due to
too many requests.')}")
        msg_box.setInformativeText(self.translate("Would you like to try scanning it again?"))

        retry_button = msg_box.addButton(self.translate("Retry"), QMessageBox.AcceptRole)
        skip_button = msg_box.addButton(self.translate("Skip"), QMessageBox.RejectRole)

        msg_box.exec_()
```

```
        if msg_box.clickedButton() == retry_button:
            return True
        elif msg_box.clickedButton() == skip_button:
            return False
        return False


    def retry_scan(self, file_path, retries=3):
        for attempt in range(retries):
            self.result_box.append(f"Retrying scan for {file_path}... (Attempt {attempt + 1}/{retries})")
            result = virustotal.upload_file(file_path)
            if "error" in result and "429" in result["error"]:
                time.sleep(1)
            else:
                self.handle_scan_result(file_path, result)
                return

        self.result_box.append(f"❌{self.translate('Failed to scan')} {file_path} {self.translate('after')} {retries}
{self.translate('attempts. Skipping file.')}")

    def update_scan_result(self, file_path, scan_result):
            self.result_box.append(f"{self.translate('Scan Results for file')}: {file_path}")
            self.result_box.append("=" * 50)
            if "error" in scan_result:
                self.result_box.append(f"{self.translate('Scan Error:')} {scan_result['error']}")
            else:
                self.result_box.append(f"{self.translate('Total checks:')}
{sum(scan_result.get(f'{category}_count', 0) for category in ['malicious', 'harmless', 'suspicious',
'undetected'])}")
                for category in ['malicious', 'harmless', 'suspicious', 'undetected']:
                    self.result_box.append(f"{category.capitalize()}: {scan_result.get(f'{category}_count', 0)}")

            self.result_box.append("\n" + "=" * 50 + "\n")
            self.result_box.ensureCursorVisible()

    def scan_single_file(self, file_path):
        retry_count = 3
        for attempt in range(retry_count):
            scan_result = virustotal.upload_file(file_path)
            if "error" in scan_result and "429" in scan_result["error"]:
                time.sleep(5)
            else:
                break
        self.update_scan_result(file_path, scan_result)
        self.update_progress_bar()

    def update_progress_bar(self):
```

```
        self.processed_files += 1
        total_files = len(self.threads)
        self.progress_bar.setValue(int((self.processed_files / total_files) * 100))

        if self.processed_files == total_files:
            if self.infected_files:
                self.result_box.append(self.translate("General result: Found") + f" {len(self.infected_files)}" +
self.translate(" infected file(s)."))

            else:
                self.result_box.append(f"{self.translate('General result: All files are safe.')}")
            self.result_box.append("=" * 50)
            self.result_box.append(f"{self.translate('Scanning complete')}")


    def display_scan_result(self, vt_result, file_path):
        self.result_box.append(f"{self.translate('Scan Results for file')}: {file_path}")
        self.result_box.append("=" * 50)
        if "error" in vt_result:
            self.result_box.append(f"{self.translate('Scan Error:')} {vt_result['error']}")
            return

        self.result_box.append(f"{self.translate('Total checks:')} {sum(vt_result.get(f'{category}_count', 0) for
category in ['malicious', 'harmless', 'suspicious', 'undetected'])}")
        for category in ['malicious', 'harmless', 'suspicious', 'undetected']:
            self.result_box.append(f"{category.capitalize()}: {vt_result.get(f'{category}_count', 0)}")

        self.result_box.append("\n" + "=" * 50 + "\n")

    def show_infected_file_dialog(self, file_path):
        msg_box = QMessageBox(self)
        msg_box.setIcon(QMessageBox.Warning)
        msg_box.setWindowTitle(self.translate("Threat Detected"))
        msg_box.setText(f"{self.translate('The file')} {file_path} {self.translate('is infected.')}")
        msg_box.setInformativeText(self.translate("Select an action:"))

        quarantine_button = msg_box.addButton(self.translate("Quarantine"), QMessageBox.AcceptRole)
        delete_button = msg_box.addButton(self.translate("Delete"), QMessageBox.DestructiveRole)
        skip_button = msg_box.addButton(self.translate("Skip"), QMessageBox.RejectRole)

        delete_button.setStyleSheet("""
        QPushButton {
            background-color: #d9534f;
            color: white;
            font-size: 14px;
            font-weight: bold;
            border: none;
            border-radius: 6px;
```

```
        padding: 6px 12px;
    }
    QPushButton:hover {
        background-color: #c9302c;
    }
    QPushButton:pressed {
        background-color: #ac2925;
    }
""")
    skip_button.setStyleSheet("""
    QPushButton {
        background-color: #7079f0;
        color: white;
        font-size: 14px;
        font-weight: bold;
        border: none;
        border-radius: 6px;
        padding: 6px 12px;
    }
    QPushButton:hover {
        background: #5b65f5;
    }
   QPushButton:pressed {
        background: #404df7;
            }
""")
    quarantine_button.setStyleSheet("""
    QPushButton {
        background-color: #f0ad4e;
        color: white;
        font-size: 14px;
        font-weight: bold;
        border: none;
        border-radius: 6px;
        padding: 6px 12px;
    }
    QPushButton:hover {
        background-color: #ec971f;
    }
    QPushButton:pressed {
        background-color: #d58512;
    }
""")
    msg_box.setStyleSheet("""
    QMessageBox {
        background-color: #EEEFF0;
        border-radius: 8px;
        font-size: 14px;
```

```
        }
        QLabel {
            color: #333;
            font-size: 14px;
        }
        QPushButton {
            background-color: #7079f0;
            color: white;
            font-size: 14px;
            font-weight: bold;
            border: none;
            border-radius: 6px;
            padding: 6px 12px;
        }
        QPushButton:hover {
            background-color: #5b65f5;
        }
        QPushButton:pressed {
            background-color: #404df7;
        }
    """)

        if self.theme in ["Dark", "Темна"]:
            msg_box.setStyleSheet("""
        QMessageBox {
            background-color: #333333;
        }
        QLabel {
            color: white;
        }
        """)

        msg_box.exec_()

        if msg_box.clickedButton() == quarantine_button:
            return "quarantine"
        elif msg_box.clickedButton() == delete_button:
            return "delete"
        elif msg_box.clickedButton() == skip_button:
            return "skip"
        else:
            return None

    def handle_infected_file_action(self, action, file_path):
        if action == "quarantine":
            self.quarantine_file(file_path)
        elif action == "delete":
            self.delete_file(file_path)
```

```
    elif action == "skip":
        self.skip_file(file_path)
    else:
        self.result_box.append(f"{self.translate('Unknown action for')} {file_path}")

def quarantine_file(self, file_path):
    self.result_box.append(f"{self.translate('☐ File moved to quarantine')}: {file_path}")
    self.move_to_quarantine(file_path)

def delete_file(self, file_path):
     os.remove(file_path)
     self.result_box.append(f"{self.translate('☐ File deleted')}: {file_path}")

def skip_file(self, file_path):
    self.result_box.append(f"{self.translate('Skipped')}: {file_path}")

def move_to_quarantine(self, file_path):
    quarantine_path = os.path.join(self.QUARANTINE_FOLDER, os.path.basename(file_path))
    os.makedirs(self.QUARANTINE_FOLDER, exist_ok=True)
    shutil.move(file_path, quarantine_path)

    file_info = {
        "file_name": os.path.basename(file_path),
        "quarantine_path": quarantine_path,
        "original_path": file_path,
        "date": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "status": "quarantined"
    }

    if os.path.exists(self.QUARANTINE_LOG):
        with open(self.QUARANTINE_LOG, "r") as f:
            log_data = json.load(f)
    else:
        log_data = []

    log_data.append(file_info)
    with open(self.QUARANTINE_LOG, "w") as f:
        json.dump(log_data, f, indent=4)

def delete_selected_file(self):
    selected_item = self.quarantine_table.currentItem()
    if selected_item:
        file_entry = selected_item.data(Qt.UserRole)
        file_path = file_entry["quarantine_path"]
        if os.path.exists(file_path):
            os.remove(file_path)
            self.result_box.append(f"{self.translate('File deleted')}: {file_path}")
            self.remove_file_from_log(file_entry)
```

```
            self.load_quarantine()
        else:
            self.show_file_not_found_message(file_path)
    else:
        self.show_no_file_selected_message()

def restore_selected_file(self):
    selected_item = self.quarantine_table.currentItem()
    if selected_item:
        file_entry = selected_item.data(Qt.UserRole)
        file_path = file_entry["quarantine_path"]
        original_path = file_entry.get("original_path")
        if os.path.exists(file_path):
            os.makedirs(os.path.dirname(original_path), exist_ok=True)
            shutil.move(file_path, original_path)
            self.result_box.append(f"{self.translate('File restored to')}: {original_path}")
            self.remove_file_from_log(file_entry)
            self.load_quarantine()
        else:
            self.show_file_not_found_message(file_path)
    else:
        self.show_no_file_selected_message()

def show_no_file_selected_message(self):
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Warning)
    msg.setWindowTitle(self.translate("No File Selected"))
    msg.setText(self.translate("Please select a file to delete or restore."))
    msg.setStandardButtons(QMessageBox.Ok)

    ok_button = msg.button(QMessageBox.Ok)
    if ok_button:
        ok_button.setStyleSheet("""
        QPushButton {
            background-color: #7079f0;
            width: 100px;
            color: white;
            font-size: 14px;
            font-weight: bold;
            border-radius: 8px;
            padding: 6px 12px;
            border: none;
        }
        QPushButton:hover {
            background-color: #5b65f5;
        }
        QPushButton:pressed {
            background-color: #404df7;
```

```
        }
    """)

    msg.exec_()

def show_file_not_found_message(self, file_path):
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Warning)
    msg.setWindowTitle(self.translate("File Not Found"))
    msg.setText(f"{self.translate('The file')} '{file_path}' {self.translate('was not found.')}")
    msg.setStandardButtons(QMessageBox.Ok)
    msg.exec_()
```