МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота магістра

ІНТЕЛЕКТУАЛЬНА ТЕХНОЛОГІЯ ДЕТЕКТУВАННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Здобувач вищої освіти гр. ІН.м-12ан/2у Данило ЧЕРНЕНКО

В.о. завідувача кафедри
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

Науковий керівник:
кандидат технічних наук, доцент,
доцент кафедри комп'ютерних наук Ігор ШЕЛЕХОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
В.о.зав.кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Черненку Данилу Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інтелектуальна технологія детектування шкідливого програмного забезпечення*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд технологій, що застосовуються для розробки інтелектуальних систем; 2) Постановка завдання й формування завдань дослідження; 3) Огляд технологій, що використовуються під час розробки інтелектуальних додаків за допомогою мови Python; 4) Проектування архітектури застосунку; 5) Розробка застосунку; 6) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд технологій, що застосовуються для розробки інтелектуальних систем		
2.	Постановка завдання й формування завдань дослідження		
3.	Огляд технологій, що використовуються під час розробки інтелектуальних додаків за допомогою мови Python		
4.	Розробка застосунку		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 44 стор., 15 рис., 13 додатків., 20 джерел.

Об’єкт дослідження — процес детектування програм-вірусів.

Мета роботи — розробка алгоритму аналізу поведінки шкідливого програмного забезпечення та імплементація у вигляді програмного забезпечення.

Методи дослідження — метод аналізу поведінки.

Результати — розроблено алгоритм та програмне забезпечення системи аналізу поведінки програмного забезпечення. Спроєктований алгоритм реалізовано у формі програмного забезпечення, створеного за допомогою мови програмування Python.

СИСТЕМА АНАЛІЗУ ПОВЕДІНКИ ПРОГРАМ-ВІРУСІВ, МАТЕМАТИЧНА МОДЕЛЬ, ІНФОРМАЦІЙНА ІНТЕЛЕКТУАЛЬНА СИСТЕМА.

ЗМІСТ

ВСТУП

Комп’ютерні віруси — один із найбільш шкідливих типів програмного забезпечення в сучасних інформаційних системах. Щороку комп’ютерні віруси причиняють шкоди розміром у декілька мільярдів доларів, спричиняють системні критичні помилки, зупиняючи великі сайти та вебзастосунки, знищуючи або модифікуючі файли, підвищуючи час відгуку. За створення та поширення шкідливих програм (в тому числі вірусів) у багатьох країнах передбачена кримінальна відповідальність. Зокрема, в Україні поширення комп’ютерних вірусів переслідується і карається відповідно до Кримінального кодексу (статті 361, 362, 363). Разом із розвитком технологій кількість різновидів вірусів зростає, що створює попит на дослідження в цьому напрямі, оскільки різні шкідливі застосунки використовують різні алгоритми та поведінку. Зважаючи на вище описану інформацію, темою переддипломної практики було обрано дослідження та проєктування програмного засобу, яке надає можливість відслідковувати сучасні комп’ютерні віруси.

1 АНАЛІЗ ПРОБЛЕМИ, ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1 Аналіз інструментів для вирішення поставленої задачі

1.1.1 Комп’ютерні віруси

Комп’ютерний вірус — це тип комп’ютерної програми, який, виконуючись, копіює себе за допомогою модифікації інших комп’ютерних програм і вживлення свого коду. Загалом комп’ютерні віруси потребують приймаючої програми. Діяльність вірусу, у більшості випадків, починається з записування власного коду до хост-програми. Коли ця програма виконується, вбудований вірус актувується першим, спричиняючи зараження та пошкодження. Деякі різновиди шкідливого програмного забезпечення, такі як комп’ютерний хробак, не потребують програми-хоста, оскільки являють собою відокремлену програму або частину коду. У такий спосіб, вони не обмежені хост-програмою, але можуть виконуватися незалежно та активно здійснювати атаки. Творці вірусів використовують знання із соціальної інженерії та експлоатують глибокі знання з властивостей безпеки для того, щоб із самого початку заразити систему та розповсюдити вірус. Віруси використовують складні стратегії поведінки для того, щоб не бути поміченими антивірусними програмними засобами. Мотивами для створення програмного забезпечення можуть бути бажання збагатитись (використовуючи програми-вимагачі), бажання надіслати якесь політичне повідомлення, персональні розваги, демонстрація наявності вразливості в програмному забезпеченні, саботаж, відмова в обслуговуванні, або просто для розслідування випадків пов’язаних із кібербезпекою, штучним життям та еволюційними алгоритмами. У відповідь на це, індустрія антивірусних програмних засобів почала швидко розвиватись, у останні роки, продаючи чи безкоштовно надаючи захист від вірусів, користувачам різних операційних систем.

Перша академічна робота, на тему теорії комп’ютерних програм, що самовідтворюються, була написана в 1949 Джоном фон Нейманом, який проводив лекційні заняття в університеті Іллінойсу про «Теорію та організацію складних

автоматичних машин». У своєму есе фон Нейман описав, як програма може бути спроєктована для того, щоб самовідтворюватися. Дизайн фон Неймана для самовідновлюваної комп'ютерної програми розглядається як перша програма-вірус, що дає привід вважати його «батьком» комп'ютерної вірусології. У 1972 Вейс Рісак опублікував статтю «Програма, що самовідтворюється, з мінімальним обміном інформації». Ця стаття описувала повноцінно функціонуючий вірус написаний на мові програмування асемблер, для комп'ютерної системи під назвою SIEMENS 4004/35. У 1980 Юрген Краус написав свою дипломну роботу «Самовідновлення програмних засобів» в університеті Дортмунда. У цій роботі Краус постулював, що комп'ютерні програми можуть поводити себе як біологічні віруси. Перший комп'ютерний вірус, типу черв'як, називався «Кріпер» і вперше був помічений у системі Arpanet, передвісниці сучасного Інтернету, на початку 1970 років. Кріпер був експериментальною програмою зі здатністю до самовідновлення, що була написана Бобом Томасом у американській дослідницькій компанії BBN Technologies у 1971 році. «Кріпер» використовував Arpanet для того, щоб інфікувати комп'ютери типу DEC PDP-10, які, зі свого боку, працювали на операційній системі TENEX. Кріпер заволодівав доступом через Arpanet та копіював себе до віддаленої системи, де повідомлення, з написом «Мене звуть Кріпер. Впіймай мене, якщо зможеш!», було показано. Згодом спеціальна програма під назвою «Жнець» для видалення «Кріпера» була створена. У 1982, програма під назвою «Elk Cloner» була першою програмою-вірусом у реальному світі, яка розповсюджувалася, навіть поміж персональних комп'ютерів, тобто поза межами комп'ютерної лабораторії, де вона була створена. Написана в 1981 році Річардом Скрентою, дев'ятикласником із вищої школи Mount Lebanon, біля Пітсбургу, вона об'єднувала себе з операційною системою Apple DOS 3.3 і розповсюджувалася через дискету. На 50-тому циклі виконнання «Elk Cloner» активувалася, заражаючи персональний комп'ютер і виводячи на екран поему, яка починалася зі слів: «Elk Cloner: програма з особистістю…». У 1984 Фред Коен з університету северної

Каліфорнії написав статтю «Комп'ютерні віруси — теорія та експерименти». Це була перша публікація, яка явно називала програму, що самовідтворюється, терміном «вірус», який був представлений наставником Коена Леонардом Адлеманом. У 1987, Фред Коен опублікував демонстрацію, що не існує такого алгоритму, який має змогу виявляти всі можливі віруси. Його теоретичний вірус був прикладом віруса, який не був шкідливим програмним забезпеченням, але був, нібито, доброзичливим (з добрими намірами). Але професіонали з розроблення антивірусів не приймають концепт «доброзичливих вірусів», оскільки, будь-який необхідний функціонал може бути реалізован без залучення вірусу (автоматичне стиснення, наприклад, доступно в операційній системі Windows, на розсуд користувача). Будь-який вірус, за визначенням, вносить несанкціоновані зміни до комп'ютера, що є небажаним, навіть, якщо ніякої шкоди не завдано або не передбачено. Перша сторінка енциклопедії вірусів доктора Соломона пояснює небажаність вірусів, навіть якщо вони нічого не роблять, крім як самовідтворення. Стаття, яка описує «корисні функції вірусів», була опублікована в 1984 Джон Баттіскомб Ґанном під назвою «Використання функцій вірусу для надання доступу користувачу до віртуального APL інтерпретатора». Перший вірус, який вражав IBM PC у реальному світі був вірусом завантажувального сектору, що представляв собою клона програми-віруса під назвою «Мозок», який був створений у 1986 Амджадом Фаруком Алвими та Басіт Фарук Алві в Лахорі, як повідомляється, для запобігання несанкціонованому копіюванню програмного забезпечення, яке вони створили. Один з перших представників програм-вірусів, які спеціально були націлені на Microsoft Windows, що мав назву «WinVir», був знайдений у квітні 1992 року, через 2 роки після випуску Windows 3.0. Вірус не містив викликів Windows API, натомість покладався на преривання DOS. Декілька років потому, в лютому 1996, австралійські ґакери з команди, яка спеціалізувалася в написанні вірусів, під назвою «VLAD», створили вірус під назвою «Bizatch» (также відомий як «Boza»), що є першим вірусом націленим на Windows 95. Наприкінці 1997, зашифрований

вірус «Win32.Cabanas» був випущений. Це перший вірус який був націлений на Windows NT (він також мав змогу заражати такі операційні системи як Windows 3.0 та Windows 9x). Навіть персональні комп'ютери були зараженими вірусами. Перший із з'явившихся на системі Commodore Amiga був вірус завантажуючого сектору, який називався «SCA», який було виявлено в листопаді 1987.

1.1.2 Будова вірусного програмного забезпечення.

Комп'ютерний вірус загалом складається з 3 частин: механізм зараження, що знаходить та заражає нові файли, триггер, що визначає, коли активувати пейлоад, пейлоад — шкідливий код, який має виконуватися.

1.1.2.1 Механізм зараження

Так званий вектор зараження, метод розповсюдження вірусу. Деякі з вірусів мають алгоритм пошуку, який знаходить та заражає файли на диску. Інші віруси заражають файли під час їхнього виконання. Прикладом такого вірусу є Jerusalem DOS вірус.

1.1.2.2 Триггер

Також відомий як логічна бомба, це частина вірусу, яка визначає умову, для якої пейлоад активується. Умова може бути точною датою, часом, наявністю іншої програми, місцем на диску, яке перевищує обмеження або відкриттям певного файлу.

1.1.2.3 Корисне навантаження або пейлоад

Корисне навантаження — це тіло вірусу, яке виконує шкідливу активність. Прикладами такої активності можуть бути: руйнування файлів, викрадення конфіденційної інформації або шпигунство за зараженою системою. Діяльність пейлоаду в деяких випадках може бути помітною, оскількі вона здатна призводити до сповільнення системи або зависання. Іноді корисні навантаження не є руйнівними, і їхня основна мета полягає в тому, щоби поширити повідомлення якомога більшій кількості людей.

1.2 Постанова завдання

Мета цієї роботи — це дослідження інтелектуальної технології виявлення шкідливого програмного забезпечення та проєктування програмного засобу. Для досягнення поставленої мети необхідно виконати наступні кроки:

1. Обрати необхідні інструменти для проектування інформаційної системи.
2. Методом аналізу підібрати найбільш підходящу математичну модель для навчання.
3. Виконати розробку вибраної моделі для машинного навчання.
4. Збір даних для моделі.
5. Виконати навчання розробленої моделі.
6. Виконати тестування інформаційної моделі

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Машинне навчання

Машинне навчання — це сфера, яка вивчає здобуття корисної інформації з даних. Машинне навчання являє собою перетин таких сучасних наук, як статистика, штучний інтелект, інформатика, прогностична аналітика, статистичне навчання. Впровадження машинного навчання в повсякденне життя стало розповсюдженою справою в наш час. Результатами робіт_цієї сфери користуються в багатьох напрямах, починаючи з рекомендацій_щодо вибору фільмів, доставки їжі та закінчуючи персоналізацією музичних уподобань та розпізнаванням знайомих людей на фотографіях. Коли людина використовує сайти зі складною вебархітектурою, наприклад, такі як Netflix, YouTube, Facebook скоріш за все кожен розділ цього сайту містить декілька моделей машинного навчання. Що стосується сфер, які не є комерційно-прибутковими, то машинне навчання використовується в астрономії для знаходження віддалених планет, у біології та медицині для аналізу послідовностей ДНК та адаптації лікування злоякісних пухлин під особливості людини. На початку розвитку «розумних» додатків багато систем використовували «hardcoded» правила, які являли собою конструкції коду, наприклад, «if», «else». Для кращого розуміння можна навести приклад: спам-фільтр, завдання якого, полягає в тому, щоб відправляти відповідні листи, що надходять користувачу, до спам папки. У цьому випадку користувач може створити список, якій містить словосполучення, які допомагають спам-фільтру визначити чи є лист спамом. Такий підхід був прикладом дизайну системи за допомогою експертного рішення, для проєктування "розумного" застосунку. Правила фільтрації, створені вручну, здебільшого є прийнятними, зокрема, у тих випадках, коли людина розуміє процес моделювання. В іншому випадку підхід із використання статично закодованих правил має два головні недоліки:

- Логіка застосунку звужує вибір варіанту рішення до певного завдання чи галузі. Навіть незначна зміна поставленої завдання, може спричинити повну реконструктуризацію системи.
- Проєктування системи правил вимагає глибокого розуміння того, як рішення повинно бути прийнято людиною експертом.

Наприклад, приведений вище підхід не може бути використаний у розпізнаванні облич людей. Але це не заважає будь якому новітньому смартфону розпізнавати обличчя користувачів. Не зважаючи на швидкий розвиток технологій, це завдання залишалося не вирішеним аж до 2001 року. Головна проблема полягає в тому, що комп'ютер сприймає набір пікселів, з яких складається зображення, відмінно від людини. Ця відмінність у сприйнятті робить майже неможливим для людини розроблення набору правил, який би унікально ідентифікував обличчя на електронному зображенні. Використання ж машинного навчання, просто представляє собою програму з великою кількістю зображень облич людей, якої достатньо для алгоритму, щоб визначити, які характеристики потрібні для розпізнавання обличчя людини.

Найбільш успішними типами алгоритмів машинного навчання є ті, які автоматизують процес прийняття рішень через узагальнення з доступних прикладів. У розглянутому вище прикладі, який несе назву «навчання під наглядом», користувач надає алгоритму пари вхідних та необхідних вихідних даних. У такий спосіб, алгоритм знаходить шлях для створення необхідного результату. У цьому випадку алгоритм має змогу створити результат за введеними даними без допомоги людини. Повертаючись до нашого приклада з класифікацією спаму, використовуючи машинне навчання, користувач забезпечує алгоритм великою кількістю електронних листів, разом з інформацією, яка повідомляє чи є листи спамом. Згодом, забезпечивши алгоритм новим листом, алгоритм вираховує припущення, чи є даний електронний лист спамом.

Алгоритми машинного навчання є одним із багатьох представників, котрі навчаються з пар даних входу/виходу, називаються керованими алгоритмами навчання, тому що «наглядач» надає правильні результати щодо прикладів, на яких навчається алгоритм. Водночас як створення набору даних для входу/виходу є трудомісткою роботою, навчання під наглядом є більш зрозумілим та є легким у вимірюванні ефективності. Якщо ваш додаток може бути переведений до форми, яка потребує навчання під наглядом і розробник зможе створити набір даних, який включає в себе бажані результати, тоді, імовірніше, що машинне навчання є вирішенням поставленого завдання.

Ми можемо привести такі приклади завдань, які вирішуються за допомогою машинного навчання під наглядом:

- Ідентифікація поштового коду, написаного вручну, на поштовому конверті. У цьому завданні вхідними даними є зображення написаного поштового коду, а бажаним результатом є реальна цифра поштового коду.
- Визначення типу пухлини на основі медичного знімка. У_цьому випадку на вхід поступає зображення, а на вихід — відповідь, чи є пухлина доброякісною або злоякісною. Також розробнику системи знадобиться експертне рішення, щодо всіх зображень пухлин.
- Визначення шахрайських фінансових транзакцій. У цьому випадку на вхід подаються дані транзакцій, а на виході отримуємо відповідь, чи була транзакція здійснена шахраями чи ні. Цю проблему можно розглядати, припустивши, що власником системи з прийняття рішень є організація, яка займається фінансовими питаннями, зберігає всі транзакції та фіксує звернення користувачів із приводу шахрайських дій.

Цікаво зазначити, що навіть якщо в усіх трьох випадках вхідні/вихідні дані здаються дуже схожими, то методи збору інформації значною мірою відрізняються між собою. Якщо зчитування поштового коду є кропіткою роботою, то принаймні воно просте та дешеве. Що стосується отримання медичних знімків та діагностики,

то вони вимагають не тільки спеціального обладнання, а й дорогої експертного оцінювання. Також, у роботі з медичними знімками може підніматися питання етики та приватної інформації. У випадку виявлення шахрайських фінансових транзакцій, набір даних, що використовується, є набагато простішим. Це обумовлено тим, що користувачі фінансової установи самостійно нададуть провайдеру послуг необхідні вихідні дані, повідомивши про крадіжку. Усе що повинен зробити провайдер, це структурувати отриману інформацію у вигляді пар вхідних/вихідних даних підтверджених шахрайських транзакцій та помилкових.

Самостійні алгоритми й алгоритми без нагляду є типом алгоритмів при використанні, яких, відомими є тільки вхідні дані. Не зважаючи на те, що є багато успішних імплементацій машинного навчання, котрі включають у себе дані алгоритми, вони є більш складними в сприйнятті та в аналізі їхньої ефективності.

Приклади реалізації алгоритмів, які використовують метод навчання без нагляду:

- Визначення тем у блоку постів. Якщо людина має велику кількість текстових даних, у такому випадку, може з'явитися необхідність узагальнити інформацію наведену в ній та знайти відповідні теми. Людина може не знати заздалегідь, які теми висвітлюються в тексті чи скільки таких тем може бути. У запропонованому випадку користувач немає вихідної інформації.
- Сегментування клієнтів на групи за схожими вподобаннями. Часто, упродовж процесу аналізу даних клієнтів, виникає необхідність з'ясувати схожість між ними та між їхніми вподобаннями. Наприклад, для вебсайту, що займається продажем клієнти можуть поділятися на_такі підгрупи: «батьки», «читачі», «геймери». У такому разі власники вебсайту не можуть знати в повною_обсязі скількі чи які групи можуть бути, тому система немає жодних вихідних даних.
- Виявлення підозрілої активності на сайті. Для знаходження багів та вразливостей вебсайтів, є дуже зручним знайти схеми поведінки, які відрізняються від норми. Кожен випадок підозрілої активності може бути унікальним і також

розробники сайту можуть взагалі не мати зафіксованих випадків підозрілої поведінки. Оскільки в цій ситуації розробник може тільки спостерігати за мережевим трафіком та не знати як виглядає нормальна та підозріла поведінка рішенням є алгоритм машинного навчання без нагляду.

Для обох алгоритмів є важливим представлення дати в тій формі, яка буде легкою для розуміння комп'ютера. У сфері машинного навчання поширеною практикою є представлення даних у вигляді таблиці. Кожна одиниця даних представляє собою рядок, а кожна властивість, яка її описує це стовпчик. У випадку сегментування клієнтів розробник може описувати кожну людину за віком, статтю, датою створення акаунту, та частотою купівль. Що стосується зображень пухлин, то іхній опис може здійснюватися за допомогою відносної кількості сірого кольору в кожному пікселі, а тип може бути визначений за допомогою аналізу їхнього розміру, форми та кольору.

У машинному навчанні кожен запис або рядок називається зразком, а стовпці, характеристики які описують ці записи — властивостями.

.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Огляд програмних засобів

Є багато інструментів для завдань, які вимагають аналізу значної кількості інформації. Наприклад, Matlab, середовище для розрахунків та аналізу, яке підтримує різні парадигми обчислення. Ця система_дає змогу маніпулювати матрицями, графічно демонструвати функції та дані, проводити імплементацію алгоритмів, розробляти графічні інтерфейси для користувачів, взаємодіяти з іншими програмними засобами. Подібними інструментами є Maple, IDL, Wolfram Mathematica, та інші. Усі ці програмні застосунки є вкрай необхідними в таких сферах як машинне навчання, глибоке навчання, наука про дані, але в нашому випадку кількість функціоналу, який надають приведені вище програмні засоби, є надлишковим. Також важливими чинниками є пропрієтарна архітектура застосунків та їхня ціна. Ми не можемо змінювати програмні засоби згідно з потребами нашої проблеми. Зважаючи на ці особливості, інструментом для вирішення поставленого_завдання було обрано середовище Anaconda. Anaconda являє собою безоплатний дистрибутив із загальнодоступним програмним кодом для мов програмування Python та R, що має на меті спростити управління та розгортання пакетів для математичних та аналітичних розрахунків. Цей дистрибутив включає в себе аналітичні пакети сумісні з такими системами як Windows, Linux та MacOS.

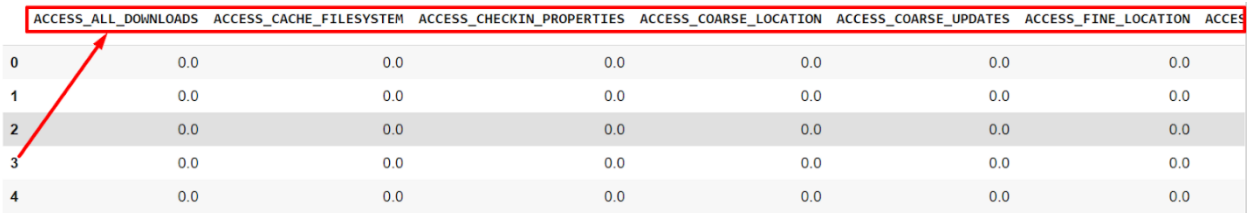
3.2 Опис вхідних даних математичної модел

Наша математична модель потребує великої кількості вхідних даних для навчання. Тому створення тестових даних вручну є дуже трудомісткою та тривалою роботою. Взявши це до уваги було вирішено використати сервіс «archive.ics.uci.edu». Цей сервіс являє собою колекцію баз даних, теорії доменів та генераторів інформації, що використовуються членами сфери машинного навчання для емпіричного аналізу. Обрані нами дані являють собою матрицю, рядки та стовпці, якої є зразками та характеристиками відповідно. Один зразок описує певне

програмне забезпечення. Що стосується властивостей, то вони описани такими атрибутами:

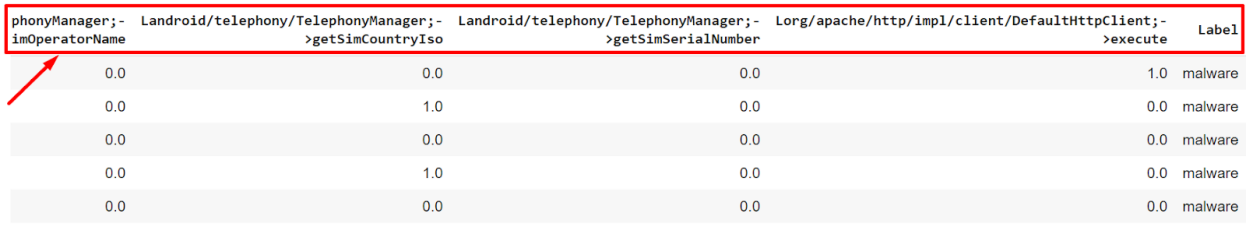
- Перші 214 атрибути, які вказують на те, чи є в програмного засобу певні права доступу.
- Властивості з 215 до 241 указують на взаємодію з API.

Завантажений набір даних має наступний вигляд (Рисунок 3.1-3.2).



	ACCESS_ALL_DOWNLOADS	ACCESS_CACHE_FILESYSTEM	ACCESS_CHECKIN_PROPERTIES	ACCESS_COARSE_LOCATION	ACCESS_COARSE_UPDATES	ACCESS_FINE_LOCATION	ACCES
0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	

Рисунок 3.1 — Початок набору даних із першими 6-ма атрибутами



phonyManager;-imOperatorName	Landroid/telephony/TelephonyManager;->getSimCountryIso	Landroid/telephony/TelephonyManager;->getSimSerialNumber	Lorg/apache/http/impl/client/DefaultHttpClient;->execute	Label
0.0	0.0	0.0	1.0	malware
0.0	1.0	0.0	0.0	malware
0.0	0.0	0.0	0.0	malware
0.0	1.0	0.0	0.0	malware
0.0	0.0	0.0	0.0	malware

Рисунок 3.2 — Кінець набору даних з останніми 5-ма атрибутами

3.3 Короткий опис програмної реалізації математичної моделі системи

Першим кроком є завантаження даних до середовища Jupyter Notebook. Після цього, за допомогою методу read_csv(), приводимо їх до форми, яка потрібна для обробки. Також ми відділяємо назву нашого класу від самих даних. Замінюємо дані типу NA/NaN на строку «goodware» та 0 відповідно. Далі ми розділяємо дані на дві групи: перша — для тренування математичної моделі, друга — для її перевірки. Після цього, починаємо процес тренування за допомогою методу fit(). Наведений лістинг програми можна побачити далі:

```
import numpy as np
import pandas as pd
```

```
from sklearn.metrics import accuracy_score
df  =  pd.read_csv('https://archive.ics.uci.edu/ml/machine-
learning-databases/00622/TUANDROMD.csv')
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
df.head(5)
target = df['Label']
target = target.fillna('goodware')
df = df.drop('Label', axis=1)
df = df.fillna(0)
X_train, X_test, y_train, y_test = train_test_split(
df, target, random_state=0)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)
y_test.describe()
accuracy_score(y_test, prediction)
test_sample = X_test.iloc[7].to_numpy()
test_sample = test_sample.reshape(1, -1)
knn.predict(test_sample)
```

Як ми можемо бачити з результату, нам успішно вдалося передбачити тип програмного забезпечення (Рисунок 3.3)

```
array(['malware'], dtype=object)
```

Рисунок 3.3 — Результат передбачення

3.4 Аналіз та розробка інформаційної системи

3.4.1 Визначення необхідних компонентів

Основною задачею нашої інформаційної системи є виявлення програм-вірусів. Тобто це означає що наша система повинна приймати на вхід певні дані чи об’єкт який харатерезуватиме програму, що перевірятимется на наявність шкідливої поведінки. Виходячи з цієї інформації одним з найпростіших рішень буде подання самого файлу програми яка має бути проаналізована. Для передачі файлу нашій системі ми будемо використовувати модуль під назвою “fileinput”. Цей модуль реалізує допоміжний клас і функції для швидкого запису циклу через стандартний ввід або список файлів. Також у програмного засобу має бути гарфічний інтерфейс для зручної взємодії користувача та інформаційної системи. Для цього будемо використовувати зовнішню бібліотеку PyQt5. PyQt — це прив’язка Python кросплатформного набору графічних інтерфейсів Qt, реалізованого як плагін Python. PyQt — безкоштовне програмне забезпечення, розроблене британською фірмою Riverbank Computing. Він доступний на тих же умовах, що й версії Qt, старші за 4.5; це означає низку ліцензій, включаючи GNU General Public License (GPL) і комерційну ліцензію, але не GNU Lesser General Public License (LGPL). PyQt підтримує Microsoft Windows, а також різні види UNIX, включаючи Linux і MacOS (або Darwin). PyQt реалізує близько 440 класів і понад 6000 функцій і методів, включаючи: значний набір графічних віджетів класи для доступу до баз даних SQL (ODBC, MySQL, PostgreSQL, Oracle, SQLite) QScintilla, віджет форматованого текстового редактора на основі Scintilla віджети з урахуванням даних, які автоматично заповнюються з бази даних парсер XML Підтримка SVG класи для вбудовування елементів керування ActiveX у Windows (лише в комерційній версії) Для автоматичного створення цих прив’язок Філ Томпсон розробив інструмент SIP, який також використовується в інших проектах. Також нам необхідно взаємодіяти з зовнішніми API систем які домоможуть в роботі нашому застосунку. Для цього будемо використовувати модуль “virustotal_python”

та “os” через які будемо взаємодіяти з системами “Windows Defender” та “VirusTotal” відповідно.

3.4.2 Проектування архітектури програмного забезпечення

Враховуючи те що нашим інструментом реалізації є мова “Python”, доцільно спроектувати систему як клас а її окремі функції та дії як методи цього класу. Назвемо наш клас “Ui_MainWindow”, та призначимо йому 2 методи під назвами “setupUi” та “retranslateUi”, які будуть керувати відображенням інформації у додатку. Відповідно полями нашого головного класу будуть елементи інтерфейсу: поля з текстом, кнопки, прапорці і т.п. Результат прокетування нашої інформаційної системи, у вигляді UML діаграми, можна побачити нижче (Рисунок 3.4).

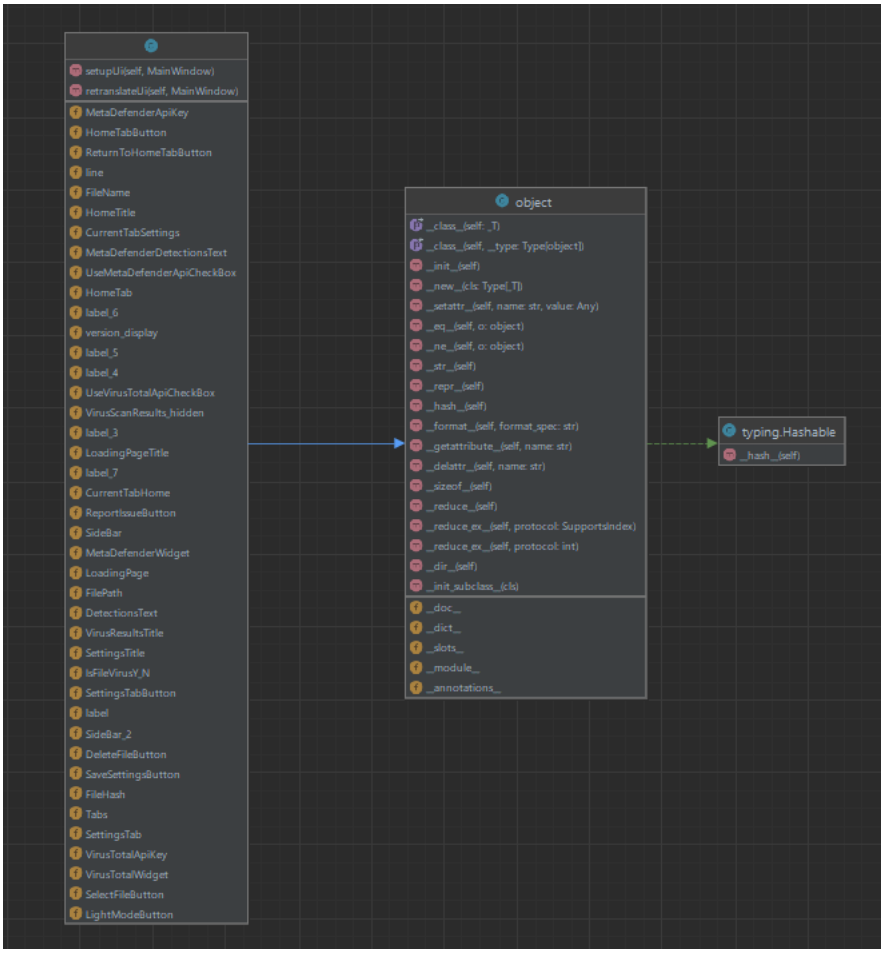


Рисунок 3.4 – UML-діаграмма нашої інформаційної системи

Як ми можемо бачити з вищенаведеної UML-діаграми, наша система є класом “Ui_MainWindow”, з відповідними полями та методами, який є нащадком класу

“object”, який в свою чергу є нащадком класу “typing.Hashable”. Впроваджений дизайн додатку можна побачити далі (Рисунок 3.4-3.6).

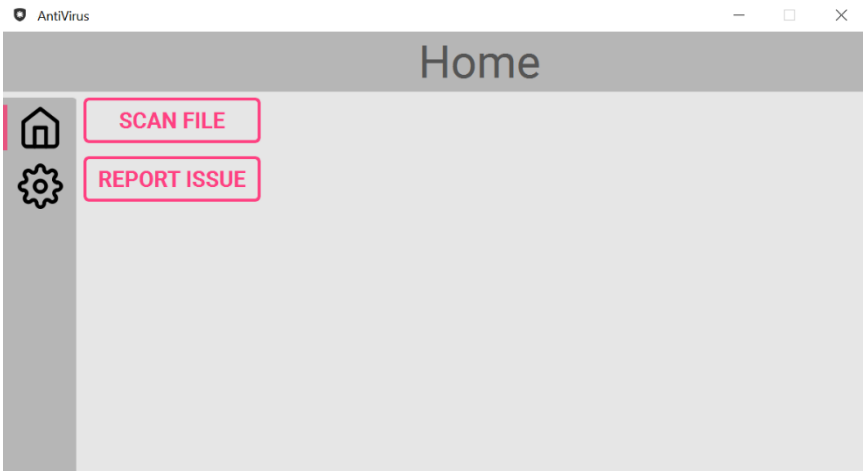


Рисунок – 3.4 Головний екран застосунку

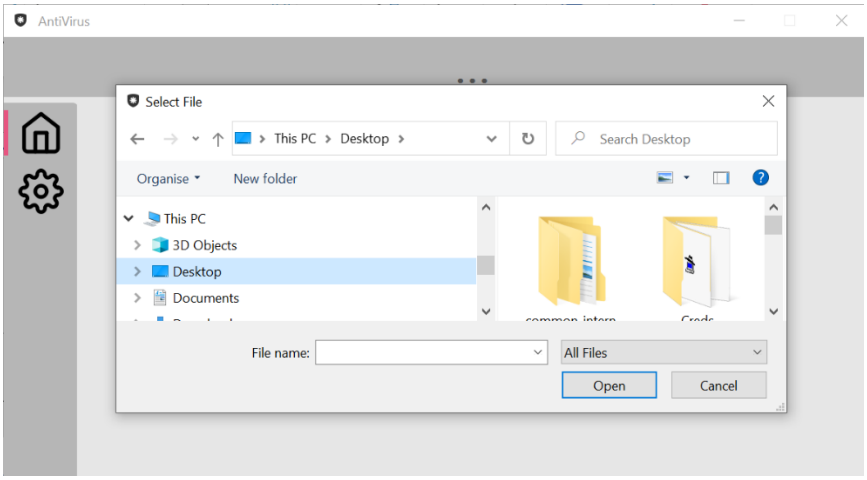


Рисунок – 3.5 Вибір фалу для аналізу, при натисненні кнопки “SCAN”

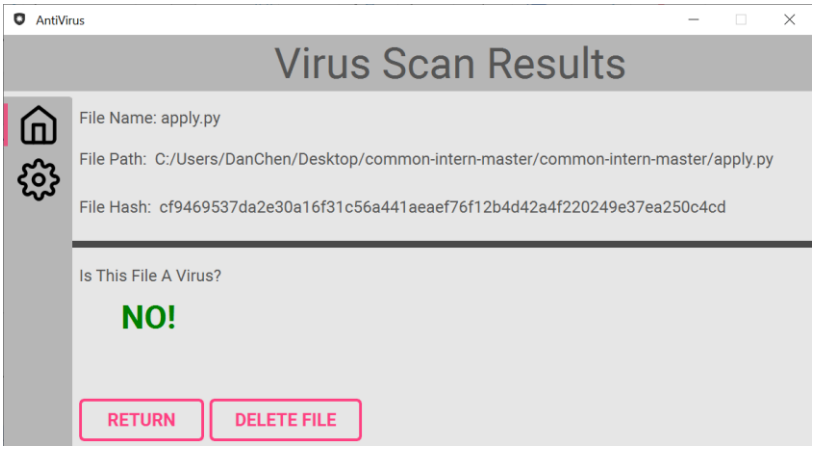


Рисунок – 3.6 Екран з результатом сканування файлу

Як ми можемо бачити з приведених вище скріншотів застосунку, ми спроектували мінімалістичний інтерфейс який виконує необхідні нам завдання.

3.5 Результати тестування інформаційної системи

При перевірці працездатності нашого застосунку було проаналізовано його основні функції. Після активації інформаційної системи було відображено вікно з головним екраном та основними елементами: кнопкою налаштування, кнопкою головного екрана, зв’язок з розробником системи через кнопку “Report issue”. (Рис. 3.7).

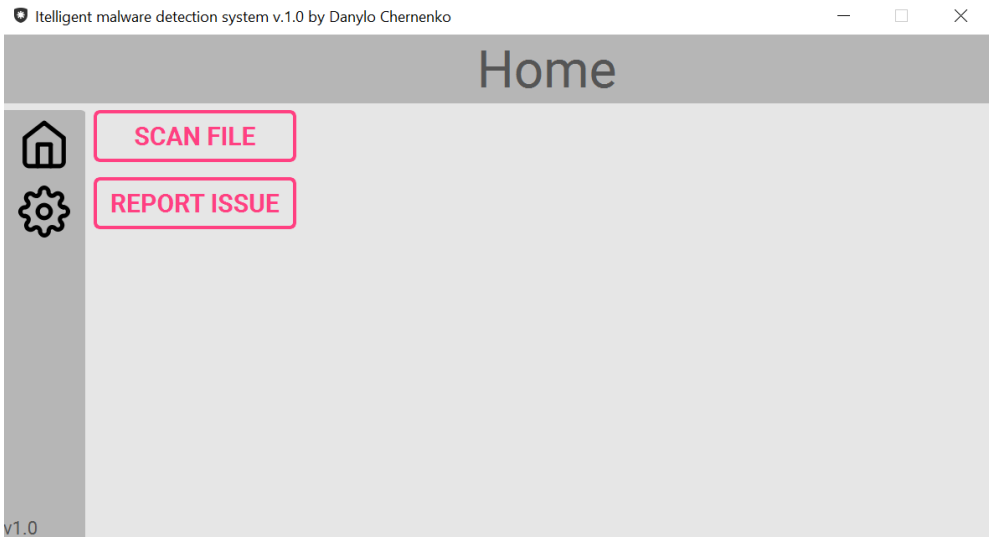


Рисунок 3.7 –Головний екран інформаційної системи

Перший етап при аналізі поведінки програмного застосунку у інформаційній системі не потребує від користувача жодних додаткових даних, крім обраного файлу для аналіза. Результат проведення аналізу інформаційної системи можана побачити на наступному зображенні (Рисунок 3.4).

Рисунок 3.8 – Результати аналізу системи

Як ми можемо бачити, згідно з результатами обрана програма не є вірусом. Для більш глибшого і точного дослідження є можливість підключення зовнішніх систем. Для того щоб підключити допоміжні системи слід ввести API ключі в необхідні поля на екрані налаштувань. Процес інтегрування з системою “VirusTotal” приведений на рис. 3.3 – 3.5.

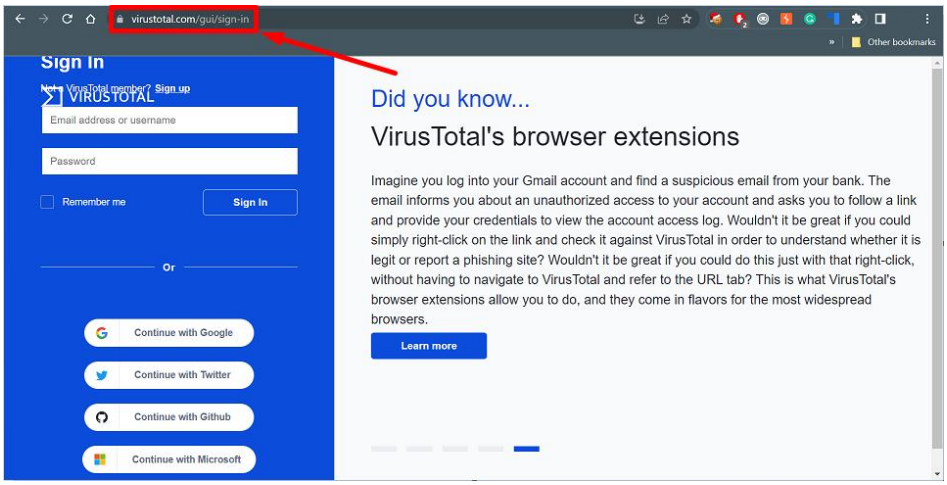


Рисунок 3.8 – Перехід до сайту “https://www.virustotal.com/en/gui”, для отримання API ключа

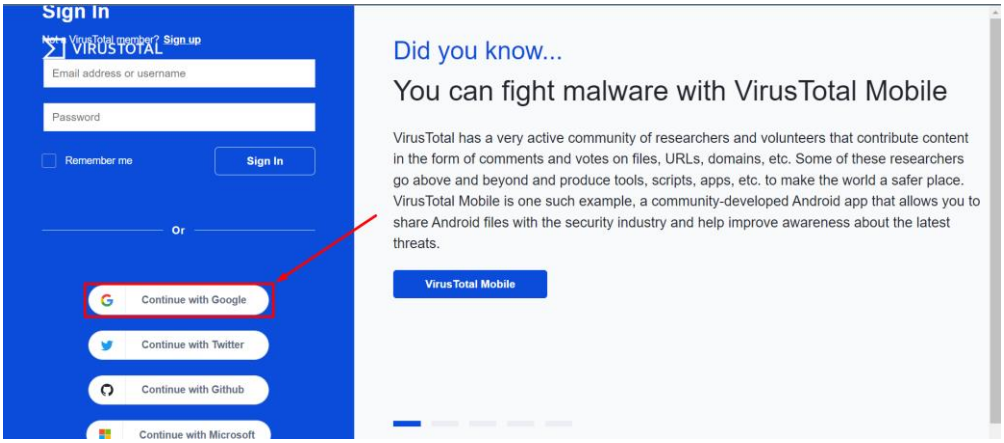


Рисунок 3.9 – Авторизація на сайті за допомогою Google SSO

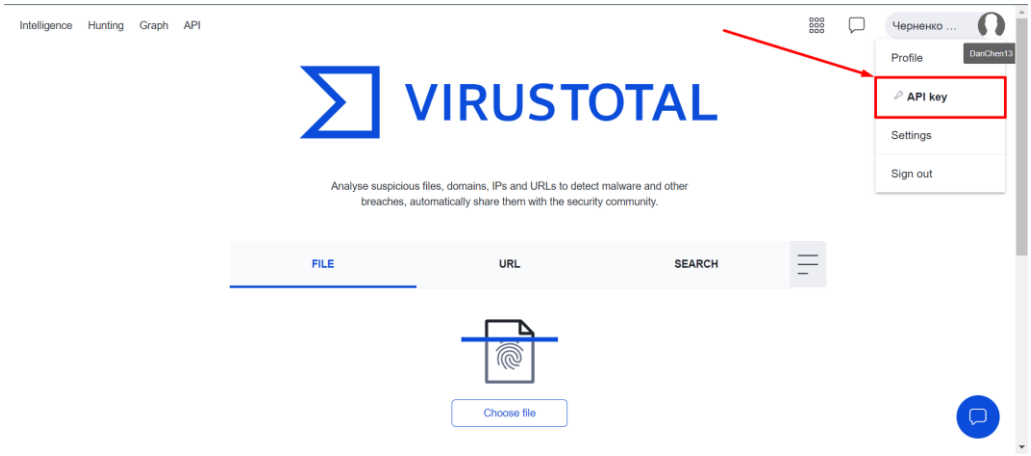


Рисунок 3.10 – Перехід до сторінки на якій знаходиться API ключ

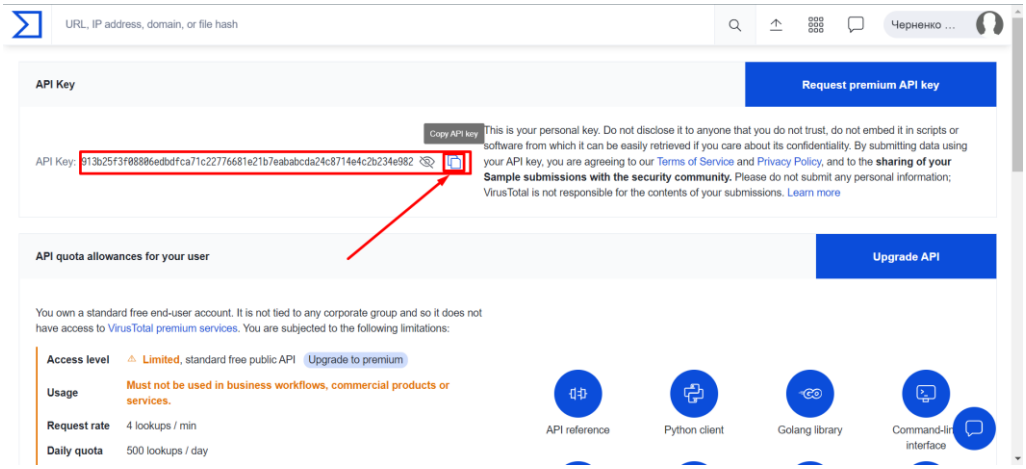


Рисунок 3.11 – Копіювання API ключа

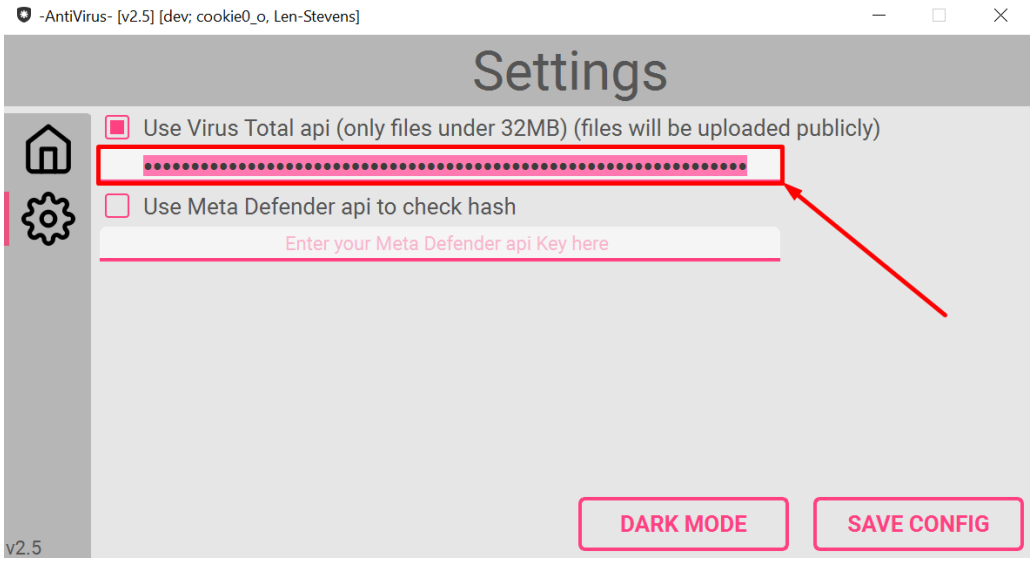


Рисунок 3.12 – Вставка API ключа у нашу інформаційну систему

В наслідок проведених операцій, тепер система може аналізувати вибрану програму або файл базуючись не тільки на нашій математичній моделі а й на інтегрованій системі. Це дозволяє розпізнавати такі віруси які значно відрізняються від тих, дані яких, ми використовували для навчання нашої моделі. Результати роботи нашої інформаційної системи після інтеграції з зовнішим ресурсом “VirusTotal” можна побачити на вищеприведеному зображенні (Рисунок 3.12).

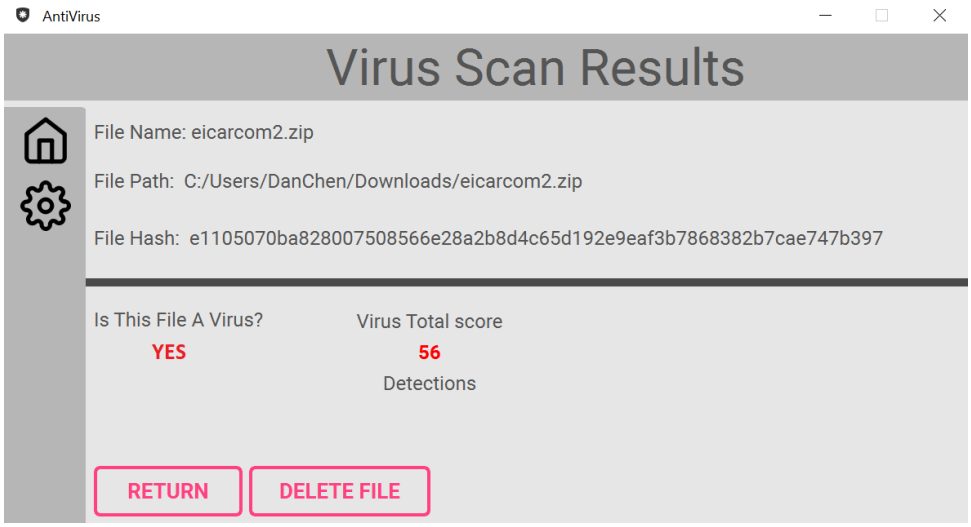


Рисунок 3.13 – Результат аналізу після інтегрування з системою “VirusTotal”

У випадку наявності шкідливої поведінки у програмного засоба чи файла, поле з відповіддю відображається червоним кольором. Якщо ж в результаті

перевірки у програми не було виявлено шкідливих властивостей, тоді поля відображаються зеленим кольором (Рисунок 3.13), вони підсвічуються червоним кольором. При необхідності коли програма все ж таки виявилася вірусом можливо використи кнопку “DELETE FILE”, що видаляє вибраний файл з операційної системи.

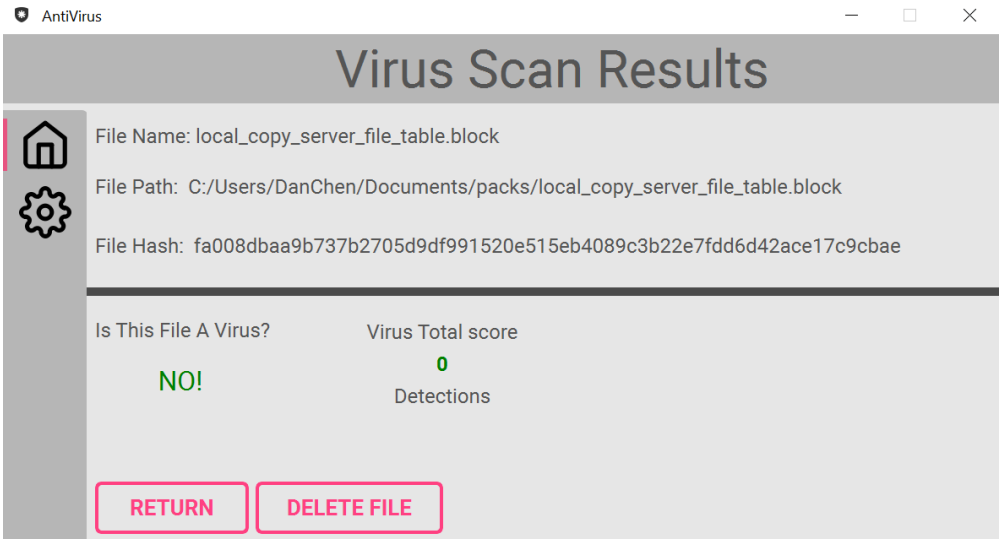


Рисунок 3.14 – Відображення полів з відповідями у випадку якщо інформаційна система не виявила шкідливих ознак програми.

При необхідності, коли програма все ж таки виявилася вірусом, для захисту системи, можливо видалити обраний файл кнопкою “DELETE FILE”.

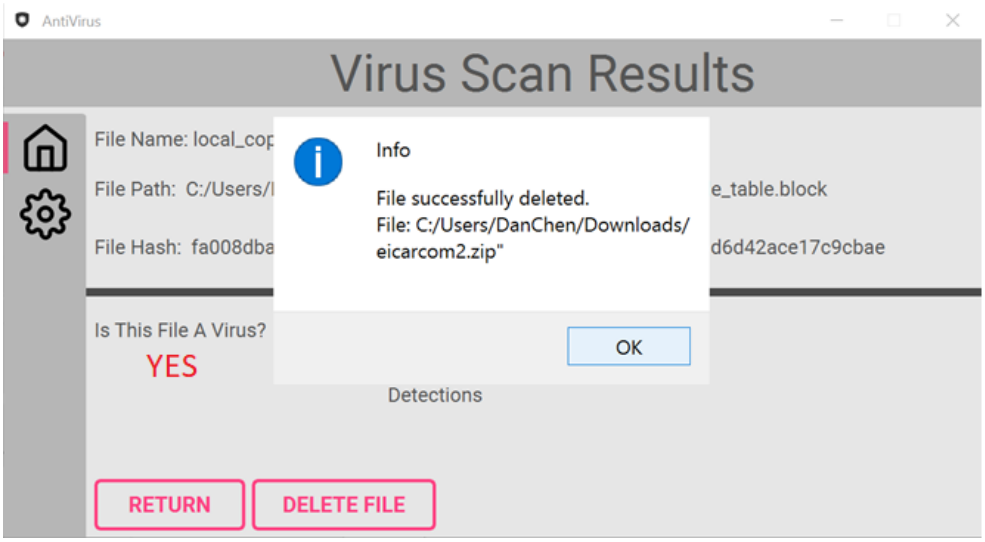


Рисунок 3.15 – Видалення програма-віруса, за допомогою кнопки “DELETE FILE”

Для того щоб повернутися на головний екран програми слід використати кнопку “RETURN”. Таким чином, спроектована інформаційна система коректно розпізнає всі сучасні програми-віруси, що використовуєтьсядля зараження комп’ютерів.

ВИСНОВКИ

В ході виконання кваліфікаційної роботи було проведено дослідження поведінки шкідливих програмних засобів або вірусів. В проведеному дослідженні використовувався метод машинного навчання. Було детально проаналізовані різні методи впровадження штучного інтелекту, технічна складова, математичне підґрунтя.

Ми успішно, виконали поставлені завдання, і довели що, використовуючи машинне навчення можливо розпізнавати шкідливе програмне забезпечення з високою імовірністю. Також для додаткової перевірки була впроваджена взаємодія з сторонніми API, а точніше з зовнішнім модулем мови Python “virustotal_python” та зовнішньою системою “Meta Defender”. Крім того, ми змогли зпроектувати та розробити програмне забезпечення, яке буде допомогати користувачу у виявлені шкідливого програмного забезпечення на його системі. Особливо треба зазначити що система була імплементована у вигляді “Frozen binary”, що дає змогу працювати на будь якій операційній системі. Це було досягнуто внаслідок формування одного пакету що містить як байт-код нашої програмної системи так і PVM (віртуальну машину Pyhton).

СПИСОК ЛІТЕРАТУРИ

1. Anaconda documentation — anaconda documentation. Anaconda Documentation — Anaconda documentation. URL: https://docs.anaconda.com/ (дата звернення: 03.12.2022).

2. Cabanas — the virus encyclopedia. Main Page — The Virus Encyclopedia. URL: http://virus.wikidot.com/cabanas (дата звернення: 25.11.2022).

3. Contributors to Wikimedia projects. Computer virus — Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Computer_virus (дата звернення: 26.11.2022).

4. Introduction to machine learning with python: a guide for data scientists. O'Reilly Media, 2016.

5. Jain D. Make a self-replicating virus in python. Medium. URL: https://infosecwriteups.com/make-a-self-replicating-virus-in-python-bb29404e3f6b (дата звернення: 27.11.2022).

6. Nokeri T. C. Data science solutions with python. Berkeley, CA: Apress, 2022. URL: https://doi.org/10.1007/978–1–4842–7762–1 (дата звернення: 29.11.2022).

7. NumPy: the absolute basics for beginners — numpy v1.23 manual. NumPy. URL: https://numpy.org/doc/stable/user/absolute_beginners.html (дата звернення: 29.11.2022).

8. Sacchetin M. Antivirus evasion with python. Medium. URL: https://infosecwriteups.com/antivirus-evasion-with-python-49185295caf1 (дата звернення: 02.12.2022).

9. 978 C. N. Rutland Water: its conception, impact and future development. Brighton: University of Sussex.

10. Bill R. Jython for java programmers. Sams, 2001. 496 p.

11. Burkov A. The hundred-page machine learning book. Andriy Burkov, 2019. 160 p.

12. Chapple M., Seidl D. CompTIA security+ study guide: exam SY0-601. Sybex, 2021.

13. Conway D. Machine learning for hackers. Sebastopol, CA : O'Reilly Media, 2012. 303 p.

14. Deep learning. MIT Press, 2017. 800 p.

15. Géron A. Hands-On machine learning with scikit-learn, keras, and tensorflow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, 2019. 856 p.

16. Hightower R. Python programming with the Java class libraries: A tutorial for building Web and Enterprise applications with Jython. Boston, MA : Addison-Wesley, 2003. 620 p.

17. Ludwig M. A. The little black book of computer viruses. Tucson, Ariz : American Eagle Publications, 1991.

18. Parikka J. Digital contagions: a media archaeology of computer viruses (digital formations). Peter Lang Publishing, 2007. 327 p.

19. Ramalho L. Fluent python: clear, concise, and effective programming. O’Reilly Media, 2021. 850 p.

20. Segaran T. Programming collective intelligence: building smart web 2.0 applications. O'Reilly Media, Inc., 2007. 360 p.

Додаток А

```
# imports
from fileinput import filename
from PyQt5 import QtCore, QtGui, QtWidgets
from virustotal_python import Virustotal
import configparser
import webbrowser
import requests
import hashlib
import sys
import os


# get current directory
current_dir = os.path.dirname(__file__)

# settings.ini file path
settings_file_path = current_dir + '/settings/settings.ini'


# define config
config = configparser.ConfigParser()
config.read(settings_file_path)

# get files with Virus hashes inside
SHA256_HASHES_pack1 = (current_dir + '\\hard_signatures\\SHA256-Hashes_pack1.txt')
SHA256_HASHES_pack2 = (current_dir + '\\hard_signatures\\SHA256-Hashes_pack2.txt')
```

Рисунок А.1 — перший рисунок додатку А

Додаток Б

```
SHA256_HASHES_pack3 = (current_dir + '\\hard_signatures\\SHA256-Hashes_pack3.txt')

# define Stuff
VERSION = "2.5"
DEV     = "cookie0_o, Len-Stevens"

# url´s
Report_issues = "https://github.com/Len-Stevens/Python-Antivirus/issues/new"
Submit_sample = "https://github.com/Len-Stevens/Python-Antivirus/discussions/8"
virus_total_api = "https://www.virustotal.com/api/v3/files/report"
meta_defender_api = "https://api.metadefender.com/v4/hash/" # + hash

# save settings to settings/settings.ini
def SaveSettings(self):
    # get api keys
    api_key = self.VirusTotalApiKey.text()
    MetaDefenderApiKey = self.MetaDefenderApiKey.text()
    # get VirusTotal scan checkbox status and meta defender scan checkbox status
    virus_total_scan = self.UseVirusTotalApiCheckBox.isChecked()
    meta_defender_scan = self.UseMetaDefenderApiCheckBox.isChecked()
    self.VirusTotalApiKey.setText(api_key)

    config['-settings-']['VirusTotalScan'] = str(virus_total_scan)
    config['-settings-']['VirusTotalApiKey'] = str(api_key)
    config["-settings-"]["MetaDefenderScan"] = str(meta_defender_scan)
    config["-settings-"]["MetaDefenderApiKey"] = str(MetaDefenderApiKey)
```

Рисунок Б.1 — перший рисунок додатку Б

Додаток В

```
    if self.LightModeButton.text() == "Light Mode":
        config["-settings-"]["Style"] = "Dark"
    else:
        config["-settings-"]["Style"] = "Light"

    with open(settings_file_path, 'w') as configfile: # save
        config.write(configfile)

    return

# removed thinker from project.
# program will now check if system is Win or Linux (if OS is Linux .ico files will not be used)

# remove file
def removeFile(file):
        try:
            os.remove(file)
        except:
            # file coudn't be deleted = show error message
            msgBox = QtWidgets.QMessageBox()
            msgBox.setIcon(QtWidgets.QMessageBox.Critical)
            msgBox.setText("Error")
            msgBox.setInformativeText(f"""\
File couldn't be deleted.
File: {file}"
            """)
```

Рисунок Б.1 — перший рисунок додатку Б

Додаток Ґ

```
            # remove window title bar
            msgBox.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
            msgBox.setWindowFlags(QtCore.Qt.FramelessWindowHint)
            msgBox.exec_()
        finally:
            # file deleted = show success message
            msgBox = QtWidgets.QMessageBox()
            msgBox.setIcon(QtWidgets.QMessageBox.Information)
            msgBox.setText("Info")
            msgBox.setInformativeText(f"""\
File successfully deleted.
File: {file}"
            """)
            # remove window title bar
            msgBox.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
            msgBox.setWindowFlags(QtCore.Qt.FramelessWindowHint)
            msgBox.exec_()


# display results
def displayResults_VIRUS(self, file):
    self.Tabs.setCurrentIndex(2)
    # check if virus total check if on and file is under 32mb
    if self.UseVirusTotalApiCheckBox.isChecked() and os.path.getsize(file) < 32000000:
        self.VirusTotalWidget.show()
    else:
```

Рисунок Ґ.1 — перший рисунок додатку Ґ

Додаток Д

```
        # hide Virus total results since it is not needed
        self.VirusTotalWidget.hide()
    # check if meta defender check if on and file is under 120mb
    if self.UseMetaDefenderApiCheckBox.isChecked() and os.path.getsize(file) < 120000000:
        self.MetaDefenderWidget.show()
    else:
        # hide meta defender results since it is not needed
        self.MetaDefenderWidget.hide()
        self.IsFileVirusY_N.setStyleSheet("color: red")
        self.IsFileVirusY_N.setText("YES!")
    # delete file button
    self.DeleteFileButton.clicked.connect(lambda: removeFile(file))
    # return button
    self.ReturnToHomeTabButton.clicked.connect(lambda: self.Tabs.setCurrentIndex(0))


def displayResults_CLEAN(self, file):
    self.Tabs.setCurrentIndex(2)
    # check if virus total check if on and file is under 32mb
    if self.UseVirusTotalApiCheckBox.isChecked() and os.path.getsize(file) < 32000000:
        self.VirusTotalWidget.show()
    else:
        # hide Virus total results since it is not needed
        self.VirusTotalWidget.hide()
    # check if meta defender check if on and file is under 120mb
    if self.UseMetaDefenderApiCheckBox.isChecked() and os.path.getsize(file) < 120000000:
```

Рисунок Д.1 — перший рисунок додатку Д

Додаток Е

```
        self.MetaDefenderWidget.show()
    else:
        # hide meta defender results since it is not needed
        self.MetaDefenderWidget.hide()
        # set text to clean
        self.IsFileVirusY_N.setStyleSheet("color: green")
        self.IsFileVirusY_N.setText("NO!")
    # delete file button
    self.DeleteFileButton.clicked.connect(lambda: removeFile(file))
    # return button
    self.ReturnToHomeTabButton.clicked.connect(lambda: self.Tabs.setCurrentIndex(0))


def scan(file, self, MainWindow):
    try:



        # default virus found to false
        virus_found = False


        # open file and get hash
        with open(file,"rb") as f:
            bytes = f.read()
            readable_hash = hashlib.sha256(bytes).hexdigest();
```

Рисунок Е.1 — перший рисунок додатку Е

Додаток Є

```
# display hash
self.FileHash.setText("File Hash:  " + readable_hash)

# check if from the selected is = to a hash in the virus hash list

# SHA256 HASHES check + pack 1
with open(SHA256_HASHES_pack1,'r') as f:
    lines = [line.rstrip() for line in f]
    for line in lines:
        if str(readable_hash) == str(line.split(";")[0]):
            virus_found = True
            f.close()
f.close()
# check if virus is found else pass
if virus_found == True:
    pass
else:
    pass
if virus_found == False:
    # SHA256 HASHES check + pack 2
    with open(SHA256_HASHES_pack2,'r') as f:
        lines = [line.rstrip() for line in f]
        for line in lines:
            if str(readable_hash) == str(line.split(";")[0]):
                virus_found = True
                f.close()
```

Рисунок Є.1 — перший рисунок додатку Є

Додаток Ж

```
f.close()
# check if virus is found else pass
if virus_found == True:
    pass
else:
    pass
if virus_found == False:
    # SHA256 HASHES check + pack 2
    with open(SHA256_HASHES_pack2,'r') as f:
        lines = [line.rstrip() for line in f]
        for line in lines:
            if str(readable_hash) == str(line.split(";")[0]):
                virus_found = True
                f.close()
    f.close()
else:
    pass
if virus_found == False:
    # SHA256 HASHES check + pack 3
    with open(SHA256_HASHES_pack3,'r') as f:
        lines = [line.rstrip() for line in f]
        for line in lines:
            if str(readable_hash) == str(line.split(";")[0]):
                virus_found = True
                f.close()
    f.close()
```

Рисунок Ж.1 — перший рисунок додатку Ж

Додаток З

```
        else:
            pass

        try:
            # check if Virus total api is checked and file is under 32mb then scan the file with Virus
            if self.UseVirusTotalApiCheckBox.isChecked() and os.path.getsize(file) < 32000000:
                # get api key
                api_key = self.VirusTotalApiKey.text()
                # check if api key is empty if yes then show error
                if api_key == "":
                    msgBox = QtWidgets.QMessageBox()
                    msgBox.setIcon(QtWidgets.QMessageBox.Critical)
                    msgBox.setText("Error")
                    msgBox.setInformativeText(f"""\
Please enter a valid Virus Total API key.
                    """)
                    # remove window title bar
                    msgBox.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
                    msgBox.setWindowFlags(QtCore.Qt.FramelessWindowHint)
                    msgBox.exec_()
                # if api key is not empty then scan the file
                else:
                    # Create dictionary containing the file to send for multipart encoding upload
                    files = {"file": (os.path.basename(file), open(os.path.abspath(file), "rb"))}

                    vtotal = Virustotal(API_KEY=api_key)
```

Рисунок 3.1 — перший рисунок додатку З

Додаток И

```
resp = vtotal.request("files", files=files, method="POST")
id = resp.data["id"]
headers = {"x-apikey": api_key}
analysis = requests.get(f"https://www.virustotal.com/api/v3/analyses/{id}", headers=headers)
analysis_json = analysis.json()
detections = analysis_json["data"]["attributes"]["stats"]["malicious"]
not_detections = analysis_json["data"]["attributes"]["stats"]["undetected"]
# show Virus total results
self.VirusTotalWidget.show()
# if detections more than half of not detections print red
if detections > not_detections:
    self.DetectionsText.setStyleSheet("color: red")
    self.DetectionsText.setText(f"{str(detections)}")
    if virus_found == False:
        self.IsFileVirusY_N.setFont(QtGui.QFont("Arial", 10))
        self.IsFileVirusY_N.setText("Probably a virus!")
    else:
        displayResults_VIRUS(self, file)
else:
    self.DetectionsText.setStyleSheet("color: green")
    self.DetectionsText.setText(f"{str(detections)}")
    if virus_found == False:
        self.IsFileVirusY_N.setStyleSheet("color: green")
        self.IsFileVirusY_N.setFont(QtGui.QFont("Arial", 12))
        self.IsFileVirusY_N.setText("Probably clean")
    else:
```

Рисунок И.1 — перший рисунок додатку И

Додаток І

```
                displayResults_VIRUS(self, file)
            else:
                pass

        # show error when virus total api was not able to scan the file
        except:
            msgBox = QtWidgets.QMessageBox()
            msgBox.setIcon(QtWidgets.QMessageBox.Critical)
            msgBox.setText("Error")
            msgBox.setInformativeText(f"""\
Cant scan file with Virus Total.
            """)
            # remove window title bar
            msgBox.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
            msgBox.setWindowFlags(QtCore.Qt.FramelessWindowHint)
            msgBox.exec_()

        try:
            # Meta Defender hash check
            if self.UseMetaDefenderApiCheckBox.isChecked():
                # get api key
                MetaDefenderApiKey = self.MetaDefenderApiKey.text()
                # check if api key is empty if yes then show error
                if MetaDefenderApiKey == "":
                    msgBox = QtWidgets.QMessageBox()
                    msgBox.setIcon(QtWidgets.QMessageBox.Critical)
```

Рисунок І.1 — перший рисунок додатку І

Додаток Ї

```
                msgBox.setText("Error")
                msgBox.setInformativeText(f"""\
Please enter a valid Meta Defender API key.
                """)
                # remove window title bar
                msgBox.setWindowFlags(QtCore.Qt.WindowStaysOnTopHint)
                msgBox.setWindowFlags(QtCore.Qt.FramelessWindowHint)
                msgBox.exec_()
            # if api key is not empty then scan the hash of the file
            else:
                M_header={"apikey": MetaDefenderApiKey}
                M_analysis = requests.get(meta_defender_api + readable_hash, headers=M_header)
                M_analysis_json = M_analysis.json()
                M_detections = M_analysis_json["scan_results"]["total_detected_avs"]
                M_not_detections = M_analysis_json["scan_results"]["total_avs"]
                half_M_not_detections = M_not_detections / 2
                # show Meta Defender results
                self.MetaDefenderWidget.show()
                # if detections more than half of not detections print red
                if M_detections > half_M_not_detections:
                    self.MetaDefenderDetectionsText.setStyleSheet("color: red")
                    self.MetaDefenderDetectionsText.setText(f"{str(M_detections)} | {str(M_not_detections)}")
                    self.IsFileVirusY_N.setStyleSheet("color: red")
                    if virus_found == False:
                        self.IsFileVirusY_N.setFont(QtGui.QFont("Arial", 10))
                        self.IsFileVirusY_N.setText("Probably a virus!")
```

Рисунок Ї.1 — перший рисунок додатку Ї