СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування

Пояснювальна записка

до магістерської дипломної роботи
магістр
(освітньо-кваліфікаційний рівень)
на тему Дослідження методів та їх практична реалізація засобу захисту комп'ютерних інформаційних систем

Виконав: студент 2 курсу, групи ІСТ-24зм
126 «Інформаційні системи та технології»
(шифр і назва спеціальності)
Дегерменджи Д. О.
(прізвище та ініціали)
Керівник Лифар В. О.
(прізвище та ініціали)
Рецензент Меняйленко О.С.
(прізвище та ініціали)

Київ – 2025 року

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ ДО МАГІСТЕРСЬКОЇ ДИПЛОМНОЇ РОБОТИ

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

Освітньо-кваліфікаційний рівень магістр

Спеціальність 126 «Інформаційні системи та технології»

(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТП
д.т.н., проф., Захожай О. І.
(підпис)
«____»_____2025р.

ЗАВДАННЯ
на магістерську дипломну роботу студенту
Дегерменджи Дмитру Олександровиу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів та їх практична реалізація засобу захисту комп'ютерних інформаційних систем
керівник роботиЛифар Володимир Олексійович, д.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу
від «08» 12 2025року №241/17.03
2. Строк подання студентом роботи 15.12.2025
3. Вихідні дані до роботи: Матеріали науково-дослідної практики, науково-методична література; дані інтернет-мережі .
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
   4.1 Вступ
   4.2 Аналіз проблеми дослідження
   4.3 Провести порівняльний аналіз існуючих антивірусних рішень.
   4.4 Виконати тестування, налагодження та оцінку ефективності роботи програмного продукту.
   4.5 Висновки
   4.6 Перелік використаних джерел
5. Перелік графічного матеріалу (з точним значенням обов'язків креслень)

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 18 жовтня 2025

КАЛЕНДАРНИЙ ПЛАН

№ з\п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Дослідження предметної галузі	25.10.2 – 28.10.25	
2	Пошук та аналіз існуючих рішень	29.10.25 – 04.11.25	
3	Аналіз проблеми дослідження	05.11.25 – 15.11.25	
5	Розробка інформаційно-аналітичної моделі	16.11.25 – 25.11.25	
6	Тестування	25.11.25 – 02.12.25	
7	Оформлення пояснювальної записки	02.12.25 – 05.12.25	
8	Підготовка та подання магістерської роботи до захисту	06.12.25 – 06.12.25	

Студент_____ Дегерменджи Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи_____ Лифар В.О.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерська дипломна робота: 68стор., 27 рис., 1 таб., 12 джерел

У ході дослідження:

створено прототип антивірусної системи з модульною архітектурою;

реалізовано сканування окремих файлів та всіх дисків системи;

впроваджено механізм потокового розподілу роботи для прискорення сканування;

реалізовано модуль оновлення бази сигнатур через завантаження даних із веб-ресурсів;

проведено тестування, у межах якого підтверджено стабільність роботи програми та коректність виконання основних функцій.

Отримані результати демонструють можливість застосування розробленого антивірусного засобу як інструмента базового захисту від загроз, а також як навчального прикладу для дослідження принципів роботи антивірусних систем.

Створений програмний продукт може бути використаний студентами, дослідниками та фахівцями як інструмент для аналізу поведінки шкідливих програм, відпрацювання навичок із кіберзахисту та моделювання антивірусних рішень. Програма є легкою для інтеграції у навчальний процес та може застосовуватись для демонстрації базових принципів сканування, оновлення сигнатур і карантинування вірусів.

Ключові слова: антивірус, комп’ютерні загрози, сканування файлів, сигнатурний аналіз, карантин, Python, інформаційна безпека.

Зміст

ВСТУП

Актуальність теми. Сучасний етап розвитку суспільства характеризується стрімкою цифровою трансформацією, у межах якої комп’ютерні інформаційні системи перетворюються на ключовий елемент функціонування підприємств, організацій, наукових установ та державних структур. Зростання залежності від цифрових технологій спричиняє підвищення вимог до їх надійності й захищеності, а питання інформаційної безпеки стають одним із найважливіших компонентів успішної діяльності на будь-якому рівні.

Кіберзлочинність, поява нових типів вірусів і шкідливих програм, атак на інфраструктуру — усе це створює реальні загрози для підприємств і користувачів. Несанкціонований доступ, викрадення конфіденційної інформації, порушення роботи операційних систем, фінансові збитки та репутаційні втрати — лише частина можливих наслідків таких атак. Тому дослідження методів захисту комп’ютерних інформаційних систем та удосконалення інструментів протидії шкідливому програмному забезпеченню є надзвичайно актуальними.

У сучасних умовах, коли кількість шкідливих програм, зокрема вірусів, хробаків, шпигунського ПЗ та інших небезпечних модулів, зростає надзвичайно швидко, важливим завданням стає впровадження ефективних механізмів контролю й аналізу даних. Одним із таких механізмів є антивірусні засоби, що забезпечують виявлення, ізоляцію та усунення шкідливого впливу на комп’ютерну систему.

Головна мета кваліфікаційної роботи полягає у дослідженні існуючих підходів до забезпечення захисту комп’ютерних інформаційних систем та розробці програмного засобу, здатного запобігати поширенню вірусів, а також виявляти та видаляти шкідливі об’єкти, що становлять небезпеку для операційного середовища.

Для досягнення поставленої мети визначені такі завдання:

дослідити класифікацію загроз комп’ютерних інформаційних систем;

проаналізувати джерела атак та механізми їх реалізації;

розглянути методи та інструменти захисту від основних видів кібератак;

здійснити порівняльний аналіз існуючих антивірусних рішень;

розробити власний програмний продукт як засіб захисту;

провести тестування створеного ПЗ та оцінити його функціональність.

Система інформаційної безпеки має бути орієнтована на такі ключові аспекти:

захист операційної системи від несанкціонованого доступу;

контроль дій користувачів, які не мають відповідних повноважень;

протидію вірусам і шкідливим програмам;

розуміння принципів роботи вірусів та механізмів їх поширення;

моніторинг мережевого середовища для виявлення підозрілої активності;

забезпечення високої доступності систем і мінімізація ризиків відмов;

підвищення рівня кіберобізнаності користувачів та персоналу.

Об’єктом дослідження є загрози та джерела небезпеки для комп’ютерних інформаційних систем.

Предметом дослідження є етапи проєктування та створення програмного засобу для протидії шкідливим програмам.

Наукова новизна роботи полягає у розробці програмного забезпечення з простим інтерфейсом та водночас ефективним набором функцій, що включає сканування окремих файлів і системи в цілому, оновлення сигнатур та карантин для ізоляції небезпечних об’єктів..

1 АНАЛІЗ ПРОБЛЕМИ ДОСЛІДЖЕННЯ

1.1 Теоретичне поняття інформаційної безпеки та комп’ютерної системи

Інформаційна безпека у сучасному розумінні розглядається як сукупність організаційних, технічних і програмних заходів, спрямованих на збереження даних від несанкціонованого доступу, неправомірного використання, пошкодження або знищення. Вона охоплює широкий спектр аспектів — від фізичного захисту приміщень до складних механізмів кіберзахисту, систем контролю доступу та політик управління ризиками. У практиці її часто асоціюють з інтегрованими рішеннями, такими як сервіси контролю доступу до хмарних ресурсів (CASB), системи виявлення аномалій, засоби протидії загрозам кінцевих точок, а також застосування принципів DevSecOps, які передбачають включення безпеки в усі етапи розробки програмного забезпечення [1].

У фундаменті інформаційної безпеки лежить так звана тріада CIA, яка визначає три ключові вимоги до будь-якої інформаційної системи та формує методологічну основу для побудови комплексної системи захисту.

1. Конфіденційність.

Цей принцип передбачає, що доступ до інформації мають отримувати виключно ті суб’єкти, яким такі права офіційно надані. Для підтримання конфіденційності застосовують методи шифрування, багатофакторну автентифікацію, контроль прав доступу та системи запобігання витоку даних [2].

2. Цілісність.

Цілісність означає забезпечення незмінності даних, їх достовірності та точності протягом всього періоду зберігання та обробки. Відповідні механізми включають контроль версій, цифрові підписи, засоби авторизації,

моніторинг змін і системи керування ідентичністю користувачів.

3. Доступність.

Цей принцип передбачає, що дані та ресурси повинні залишатися доступними для уповноважених користувачів у потрібний момент часу. Для цього необхідні регулярні оновлення, технічне обслуговування, резервне копіювання та забезпечення відмовостійкості інфраструктури.

Окрім основних складових тріади, важливими елементами системи інформаційної безпеки є:

ідентифікація та автентифікація, що гарантують коректне встановлення особи користувача;

криптографічний захист даних під час зберігання і передавання;

політики безпеки, які регламентують роботу з інформаційними ресурсами;

аналіз та оцінка ризиків, що дозволяють передбачити можливі загрози й оптимально розподілити ресурси;

засоби штучного інтелекту, які забезпечують автоматичне виявлення аномалій і реагування на інциденти в реальному часі;

планування дій у разі інцидентів, що забезпечує швидке відновлення працездатності системи.

У постіндустріальному суспільстві комп’ютерні інформаційні системи перетворилися на ключовий елемент будь-якої інфраструктури. Вони охоплюють апаратні компоненти (процесор, пам’ять, мережеві пристрої), програмне забезпечення (операційні системи, прикладні програми), а також масиви даних, які формують основу інформаційних потоків організації. Функціонування таких систем визначає продуктивність бізнесу, ефективність наукових досліджень та рівень технологічного розвитку суспільства.

Комп’ютерні системи, як складні багаторівневі структури, включають:

апаратне забезпечення, яке відповідає за обчислення, зберігання інформації та взаємодію з периферійними пристроями;

системне та прикладне програмне забезпечення, що виконує управління ресурсами та реалізує бізнес-функції;

системи управління базами даних, які забезпечують організацію, цілісність і доступність інформаційних масивів;

мережеву інфраструктуру, що забезпечує обмін даними між компонентами системи.

Усі ці складові можуть стати об’єктом атак або джерелами загроз. Джерела небезпеки поділяють на внутрішні та зовнішні, технічні, програмні, організаційні та соціальні. Вони можуть проявлятися у вигляді спроб несанкціонованого доступу, проникнення шкідливих програм, змін у конфігурації системи, викрадення даних або знищення її компонентів. Особливо критичними є загрози, пов’язані з людським фактором — помилки персоналу та недотримання правил безпеки часто спричиняють навіть більші втрати, ніж зовнішні атаки.

Таким чином, інформаційна безпека є комплексною дисципліною, що об’єднує технологічні, організаційні, правові та поведінкові механізми. Її ефективність залежить від узгодженості всіх компонентів системи та здатності організації своєчасно реагувати на сучасні види кіберзагроз.

У сучасних комп’ютерних інформаційних системах різноманіття потенційних загроз є настільки широким, що для їх ефективного аналізу застосовують багаторівневі класифікації. Вони дозволяють систематизувати загрози, визначити їх походження, характер впливу, умисність дій та умови реалізації. Знання таких класифікацій є передумовою для побудови комплексної системи захисту.

Однією з базових ознак є походження загроз, за якою виділяють природні та штучні (спричинені діяльністю людини) загрози [8]. Природні загрози викликаються стихійними явищами або фізичними процесами, що не залежать від людського фактора. Штучні ж напряму пов’язані з людськими діями, випадковими або умисними.

За рівнем вмотивованості дій загрози поділяють на випадкові та навмисні.

До випадкових належать інциденти, спричинені помилками персоналу, технічними збоями чи недбалим ставленням до процедур безпеки. Хоча такі ситуації не мають злого наміру, їх наслідки можуть бути надзвичайно серйозними — за статистикою до 80% втрат інформації пов’язані саме з неумисними діями [3].

Навмисні загрози виникають у результаті цілеспрямованих дій зловмисника, спрямованих на пошкодження, викрадення чи модифікацію інформації.

До прикладів випадкових загроз належать:

• помилки технічного персоналу;

• неналежне зберігання резервних копій або архівів;

• випадкове видалення чи зміна даних;

• раптові перебої електроживлення;

• пошкодження кабельної інфраструктури;

• вихід з ладу обладнання (серверів, мережевих адаптерів, робочих станцій);

• збої в програмному забезпеченні;

• зараження вірусами через необережне використання носіїв;

• отримання конфіденційної інформації третіми особами через помилки працівників [3].

Часто найбільші збитки виникають саме через низьку обізнаність або недотримання правил безпеки користувачами. Тому питання розмежування прав, правильної конфігурації доступу та регулярного навчання персоналу є критично важливими [4].

Класифікація навмисних загроз

До цілеспрямованих загроз, що здійснюються зловмисниками,

належать:

1. Методи промислового чи цифрового шпигунства (підслуховування, перехоплення, крадіжка носіїв, візуальне спостереження, шантаж чи підкуп співробітників).

2. Несанкціонований доступ через недосконаліть систем автентифікації або налаштувань.

3. Використання електромагнітного випромінювання апаратури для зчитування інформації.

4. Модифікація системних або програмних структур.

5. Інфікування шкідливим ПЗ різних типів [3].

Класифікація за джерелом виникнення

Загрози поділяють на природні (магнітні бурі, пожежі, повені, радіаційне випромінювання) та людські, що можуть включати як внутрішні, так і зовнішні фактори.

До людських джерел належать:

впровадження сторонніх агентів у штат;

підкуп співробітників;

зловмисне видалення або копіювання інформації;

компрометація паролів, ключів та інших атрибутів доступу.

Загрози, пов’язані з програмно-апаратними засобами

Вони виникають у разі:

невірного використання службових програм, що може призвести до зависань, втрати даних або некоректної роботи ОС;

відмов обладнання;

інсталяції неліцензійних програм, які використовують ресурси системи або несуть приховану шкідливу функціональність;

зараження вірусами та іншими шкідливими компонентами [1].

Класифікація за зоною походження загроз

1. Зовнішні загрози — спроби проникнення ззовні периметра безпеки:

перехоплення випромінювань;

аналіз трафіку;

дистанційне спостереження.

2. Загрози всередині контрольованої зони — дії зловмисників, що мають фізичний доступ до приміщення:

крадіжка документів або носіїв;

вплив на системи охолодження, електроживлення;

встановлення підслуховуючих пристроїв [5].

3. Загрози через периферійні пристрої — підключення сторонніх носіїв, підроблених USB-пристроїв, зовнішніх модемів.

4. Внутрішні загрози — пов’язані з роботою самої системи:

помилки конфігурації;

використання вразливого ПЗ;

дії незадоволених співробітників.

Класифікація за взаємозв’язком з активністю системи

Загрози, що не залежать від активності КС (крадіжка носіїв, криптоаналіз).

Загрози, що проявляються лише під час обробки даних (поширення вірусів) [25].

Пасивні та активні загрози

Пасивні — не змінюють структуру системи (перехоплення даних).

Активні — модифікують або руйнують інформацію.

Прикладами активних загроз є:

• впровадження апаратних закладок і вірусів;

• зміна режимів роботи пристроїв;

• навмисна модифікація інформації.

Класифікація за способом отримання доступу

1. Через стандартні шляхи доступу:

крадіжка паролів (у тому числі за допомогою перехоплювачів);

атака типу «маскарад» — дії від імені іншого користувача;

використання викрадених реквізитів автентифікації.

2. Через нестандартні канали:

завантаження системи з зовнішніх носіїв;

використання недокументованих можливостей ОС.

Класифікація за розташуванням інформації

Загрози можуть стосуватися:

зовнішніх носіїв (ризик копіювання);

оперативної пам’яті (зчитування залишкових даних);

каналів зв’язку (підміна пакета, несанкціоноване підключення);

відображеної інформації (зйомка екрану, перехоплення друку) [35].

Приклади основних типів атак

Серед найпоширеніших атак на автоматизовані системи:

1. Віддалене проникнення (NetBus, BackOrifice).
2. Локальна ескалація доступу (GetAdmin).
3. Відмови в обслуговуванні через мережу (Teardrop, trin00).
4. Локальна відмова — перевантаження ресурсів.
5. Сканування мереж (nmap).
6. Використання сканерів уразливостей (Nessus, Xspider).
7. Злам паролів (L0phtCrack, Crack, AZPR).
8. Прослуховування мережевого трафіку (tcpdump, Network Monitor).

1.2 Комп’ютерні віруси – вид загрози комп’ютерних інформаційних систем

Комп’ютерні віруси становлять один із найпоширеніших і найнебезпечніших різновидів загроз комп’ютерним інформаційним системам. Під терміном «вірус» розуміють різновид шкідливого програмного забезпечення, основною метою якого є несанкціонована зміна, пошкодження або знищення даних, що зберігаються чи обробляються комп’ютером.

Ураженню піддаються як виконувані програми, так і документи, фотофайли чи інші типи даних, що може спричинити часткову втрату інформації або повну неможливість подальшої роботи з файлами .

Одним із найпоширеніших різновидів шкідливих програм є файлові віруси. Вони вбудовуються у виконувані файли, модифікуючи їх повністю або частково, або ж доповнюючи їх власним кодом. Під час запуску інфікованої програми операційна система розпізнає вірус як частину легітимного застосунку, надаючи йому ті самі права доступу. Завдяки цьому вірус непомітно розмножується, створює власні копії, змінює параметри системи та залишається у пам’яті протягом тривалого часу. Через свою особливість маскуватися під звичну програму такі віруси часто називають паразитичними.

Поведінка файлових вірусів зазвичай підпорядкована певним фазам:

фаза прихованої дії, коли шкідливий код не проявляє себе і намагається уникати виявлення;

стадія розмноження, що активується після першого запуску зараженого файла та призводить до створення нових копій вірусу;

тригерна фаза, у якій активується механізм переходу до шкідливих дій;

фаза ураження, коли відбуваються руйнівні операції — від виведення небажаних повідомлень до масового видалення файлів або порушення роботи ОС .

Ще один значущий тип загроз — макровіруси, що створюються за допомогою макромов, вбудованих у прикладні пакети (Microsoft Word, Excel тощо). На відміну від файлових, середовищем їх виконання є не операційна система, а сам додаток, який обробляє макрокоманди. Макровіруси поширюються переважно через документи, вкладені до електронних листів або завантажені з ненадійних інтернет-ресурсів. Особливість таких вірусів полягає в тому, що вони активуються лише після запуску певного макросу та здатні заражати інші файли під час їх відкриття .

До окремої групи належать завантажувальні віруси. Вони уражають сектор завантаження гнучких або жорстких дисків. Після первинного старту з інфікованого носія шкідливий код активується під час кожного завантаження системи, що ускладнює його видалення. Значна частина антивірусних інструментів не може очистити головний завантажувальний запис (MBR) у робочій системі, тому часто потрібні спеціальні аварійні засоби. Такі віруси здатні спричиняти помилки читання диска, сповільнення роботи або навіть повну неможливість запуску ОС, зокрема появу повідомлення «Недійсний системний диск» .

Особливо небезпечними вважають поліморфні віруси — різновид шифрувальних вірусів, які змінюють власний код при кожному зараженні. Їхня особливість у тому, що вірус генерує різні варіанти дешифрувальних процедур, застосовує різні ключі та алгоритми шифрування. Через це традиційні сигнатурні методи захисту часто стають неефективними, оскільки кожна нова копія вірусу виглядає інакше. Поширення таких загроз відбувається зазвичай через електронну пошту, заражені вебсторінки або встановлення неперевірених програмних продуктів .

До сучасних форм шкідливого ПЗ належать також віруси-шифрувальники (ransomware). Вони проникають у систему, шифрують важливі файли користувача, а потім вимагають викуп за їх розблокування. Як правило, механізм зараження запускається після відкриття шкідливого документа або переходу за небезпечним посиланням. Після перезавантаження ОС вірус активується та блокує доступ до даних, шифруючи документи, фото, медіафайли та інші об’єкти. Сучасні варіанти таких програм вражають не лише Windows-системи, а й Linux, macOS та Android .

Завершує класифікацію комп’ютерний хробак — вид шкідливого ПЗ, що самостійно поширюється мережею без участі користувача. Хробаки активно використовують уразливості операційних систем і мережевих

сервісів, здатні швидко створювати велику кількість власних копій, уповільнюючи роботу мережі та викликаючи значні відмови у функціонуванні інфраструктури. Часто такі віруси слугують основою для подальших атак, зокрема встановлення троянських модулів чи шпигунського ПЗ.

Таким чином, комп’ютерні віруси становлять широкий спектр загроз — від порушення окремих файлів до повного паралічу системи. Їх здатність маскуватися, видозмінювати власний код, поширюватися різними каналами та активно взаємодіяти з операційною системою робить їх однією з найсерйозніших проблем у сфері інформаційної безпеки. Саме тому ефективні засоби виявлення та протидії шкідливому ПЗ є ключовим елементом захисту комп’ютерних інформаційних систем.

1.3 Антивіруси та алгоритми розпізнавання вірусів

Антивірусне програмне забезпечення є одним із ключових інструментів захисту комп’ютерних інформаційних систем від шкідливих програм. Після запуску операційної системи антивірус одразу активується та починає аналізувати файли, процеси і елементи ОС. Така робота потребує значних обчислювальних ресурсів, оскільки здійснюється постійне порівняння фрагментів коду з відомими зразками шкідливих програм, що інколи може спричиняти помітне уповільнення роботи комп’ютера. Саме з цього приводу сучасні розробники антивірусів намагаються оптимізувати архітектуру своїх продуктів так, щоб забезпечити максимальний рівень безпеки при мінімальному навантаженні на систему .

Одним із основних підходів до виявлення шкідливих програм залишається signature detection — сигнатурний аналіз. Цей метод полягає в ретельному порівнянні вмісту файлів і програмних компонентів із наборами сигнатур, які попередньо завантажені в антивірусну систему. Сигнатура — це унікальна послідовність бітів або кодових фрагментів, що характерні для

певного виду вірусу. Під час звичайної роботи антивірус перевіряє як файли, що вже зберігаються в пам’яті комп’ютера, так і ті, що щойно потрапили до системи, наприклад через завантаження з мережі чи зйомних носіїв. Якщо виявлено збіг між кодом файлу та сигнатурою з бази даних, підозрілий об’єкт автоматично переміщується в карантин — ізольовану область, де він не може взаємодіяти із системою та іншими файлами .

Алгоритм сигнатурного пошуку складається з кількох взаємопов’язаних етапів:

1. Сканування файлової системи та відбирання об’єктів для перевірки.

2. Порівняння отриманих даних зі зразками з бази сигнатур.

3. Виявлення збігу та негайна реакція — блокування, карантин або видалення.

4. Оновлення бази сигнатур, яке дає можливість розпізнавати нові види загроз.

Попри ефективність сигнатурного методу, він має й обмеження: антивірус не здатен розпізнати новий вірус, якщо його сигнатура ще не додана до бази. Це стимулювало розвиток інших методів аналізу.

Одним із таких підходів є евристичне виявлення — метод, що дозволяє аналізувати поведінку файлів або їхню структуру, навіть якщо конкретної сигнатури загрози в базі ще не існує. У комп’ютерних науках евристичний алгоритм визначають як механізм пошуку розв’язку, що наближений до оптимального. Такі алгоритми можуть швидко знаходити підозрілі дії, які характерні для вірусів: саморозмноження, створення копій, спроби модифікувати системні файли, доступ до критичних процесів тощо. Хоча результат евристичного аналізу не завжди є безпомилковим, він дозволяє виявляти загрози на ранніх етапах, коли вони ще не описані у базах сигнатур. Тому евристичні методи є важливою складовою сучасних антивірусних рішень, забезпечуючи додатковий рівень безпеки для користувача .

Прикладом евристичного підходу є жадібні алгоритми, що оцінюють

поведінку файлу за певними правилами та роблять висновок про можливу небезпеку, не аналізуючи усі можливі варіанти дій. Такі алгоритми можуть визначати підозрілі патерни у коді, відстежувати аномальні операції та застосовуються як у статичному, так і у динамічному аналізі програм.

У поєднанні з сигнатурними та евристичними методами сучасні антивіруси можуть використовувати й інші технології — поведінкове спостереження, машинне навчання, аналіз активності у реальному часі. Проте для базового рівня захисту сигнатурний та евристичний методи залишаються фундаментальними, забезпечуючи ефективне виявлення більшості відомих та нових шкідливих програм.

1.4 Порівняльний аналіз існуючих антивірусних програм

Під час аналізу антивірусного програмного забезпечення важливо комплексно враховувати низку характеристик, що визначають загальний рівень захисту. До основних критеріїв належать: ефективність виявлення загроз, зручність використання та надійність механізмів захисту. Саме ці показники дозволяють об’єктивно оцінити роботу антивірусів і визначити їх переваги та недоліки в реальних умовах.

Для порівняння було обрано п’ять популярних антивірусних програм:

Zillya! Total Security, Avast Free AiVirus, Bitdefender Internet Security, Avira Internet Security та Windows Defender. Кожен із цих продуктів має власні підходи до побудови системи захисту, різний набір функцій, рівень продуктивності та особливості взаємодії з користувачем, що й зумовлює необхідність їх комплексного аналізу.

Zillya! Total Security

Zillya! — український антивірус, орієнтований на користувачів різних категорій: домашніх, корпоративних та мобільних. Продукт має велику базу сигнатур — понад 15 мільйонів зразків шкідливих програм, що забезпечує

широкий спектр можливостей для виявлення загроз. Серед переваг Zillya! — зручний інтерфейс, наявність різних режимів сканування й регулярні оновлення баз.

Недоліками користувачі найчастіше називають можливі хибні спрацювання та нестабільність роботи системи після видалення або лікування частини заражених файлів. Також зазначається, що програма інколи має труднощі з розпізнаванням складних видів вірусів.

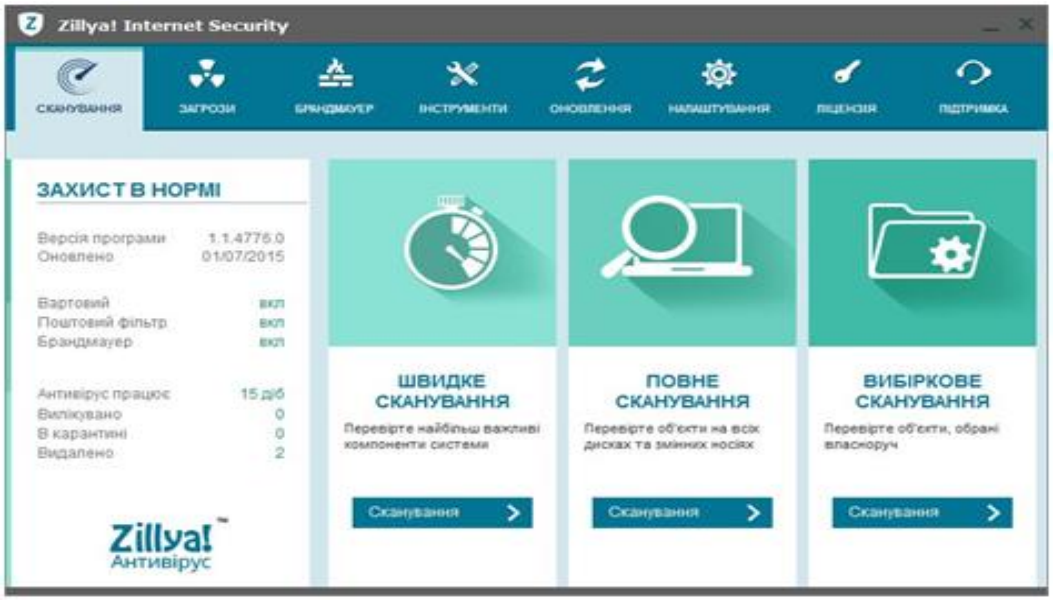

Рисунок 1.1 – Zillya! Антивірус

Avast Free AiVirus

Avast — один із найпопулярніших безкоштовних антивірусів. Він містить широкий набір інструментів, зокрема перевірку електронної пошти, моніторинг підозрілої активності програм, сканування Wi-Fi-мереж, захист від шифрувальників та онлайн-загроз. Також існують розширені платні версії.

Avast вирізняється високою швидкістю сканування, гнучкістю налаштувань і простотою використання. Проте у безкоштовній версії інтерфейс частково обмежений і можуть виникати сповіщення рекламного

характеру. У деяких випадках програма створює помітне навантаження на систему.

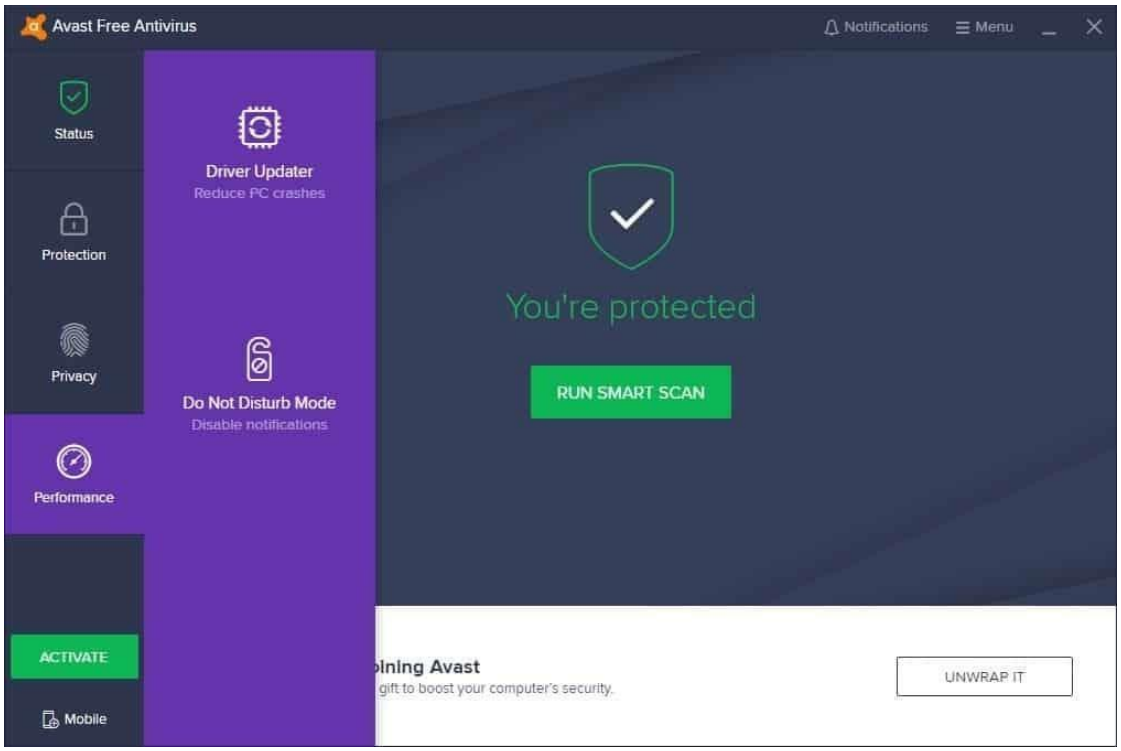

Рисунок 1.2 – Avast Free AiVirus

Windows Defender

Windows Defender — вбудований антивірус операційної системи Windows, який автоматично активується після встановлення ОС. Тісна інтеграція із системою забезпечує стабільну роботу та мінімальне навантаження на ресурси ПК. Defender оперативно отримує оновлення, використовує поведінковий аналіз і механізми хмарного захисту.

Згідно з дослідженнями, Windows Defender має високі показники ефективності, зручності та загальної якості захисту, і часто не поступається стороннім комерційним рішенням.

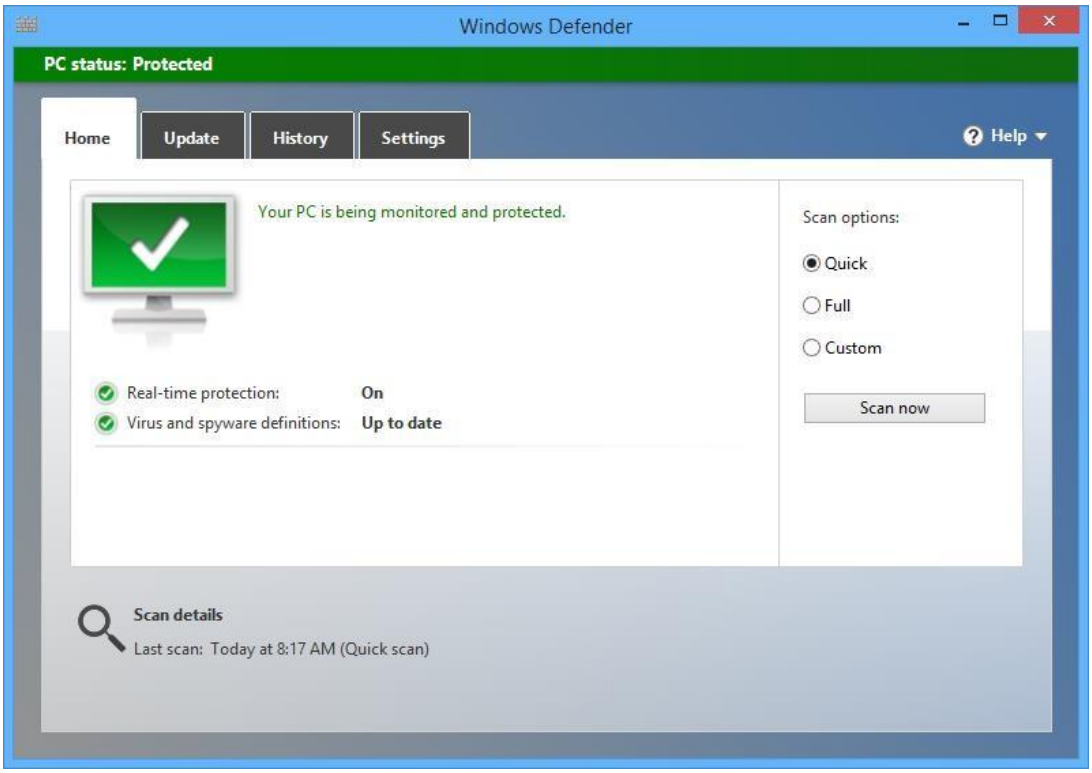

Рисунок 1.4 – Windows Defender

Bitdefender Internet Security

Bitdefender отримав найвищі оцінки серед усіх протестованих програм. Антивірус використовує багаторівневий захист, включно з поведінковим аналізом, технологіями запобігання фішингу та фільтрацією спаму. Незважаючи на значні вимоги до апаратних ресурсів, програма забезпечує стабільну роботу й найкращі результати у виявленні складних загроз.

За результатами порівняльної таблиці Bitdefender продемонстрував найвищу загальну ефективність із усіх представлених антивірусних рішень.

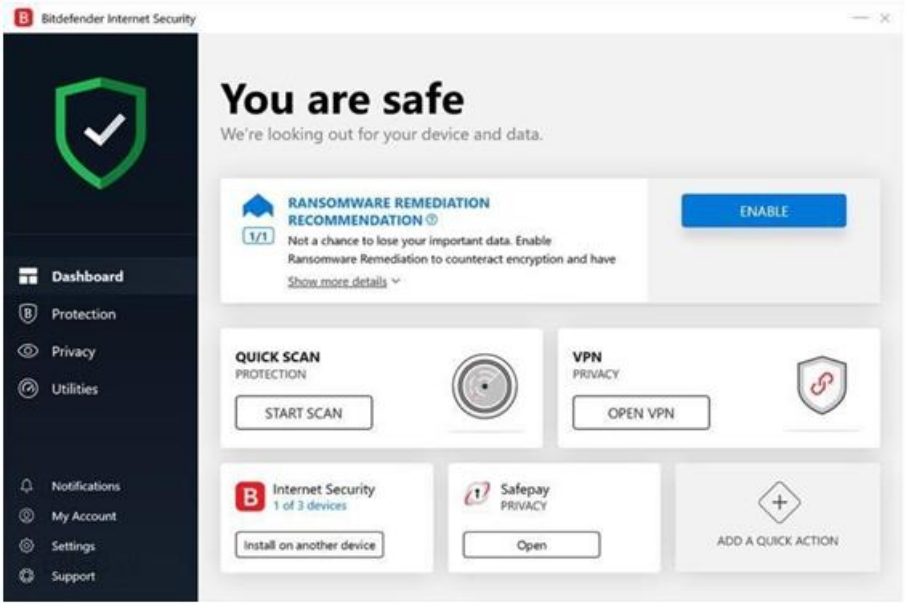

Рисунок 1.5 - Bitdefender Internet Security

Avira Internet Security

Avira має високу швидкість сканування, широкий набір захисних інструментів та ефективний механізм протидії програмам-вимагачам. Антивірус також забезпечує захист вебперегляду, контроль доступів і фільтрацію підозрілих інтернет-ресурсів.

Недоліками Avira є підвищене навантаження на системні ресурси та періодичні хибні спрацювання. Деякі користувачі також повідомляють про труднощі зі службою підтримки.

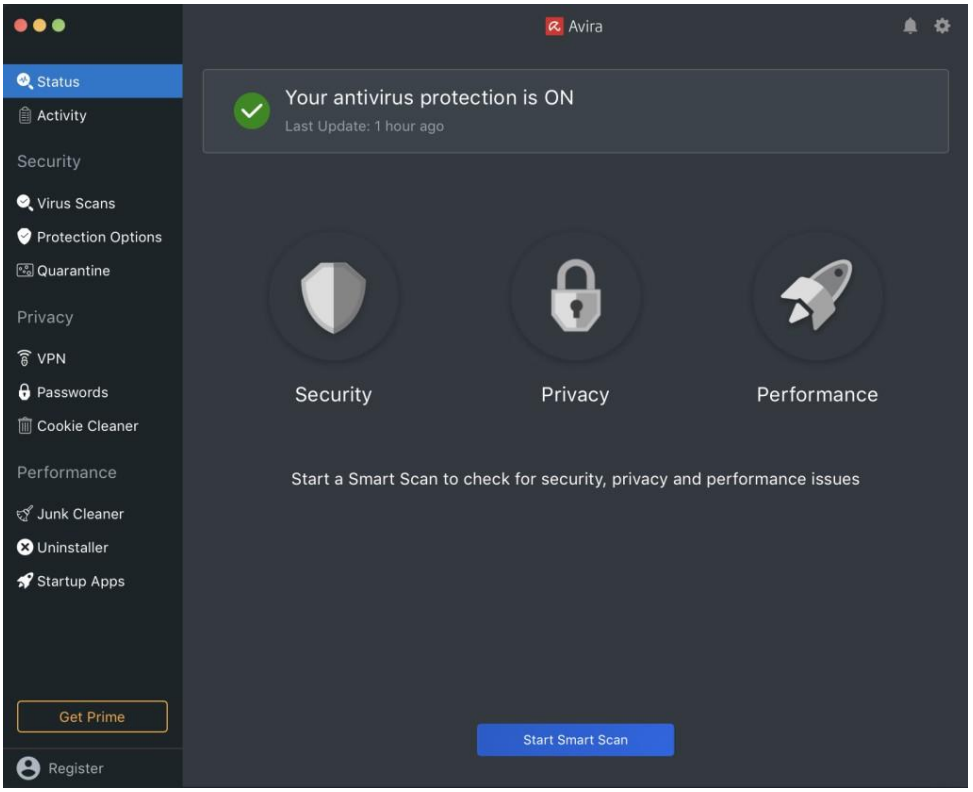

Рисунок 1.6 – Avira Internet Security

Підсумок аналізу

Усі досліджувані програмні продукти були протестовані на різних видах шкідливих файлів і оцінені за трьома критеріями:

ефективність виявлення,

зручність роботи,

рівень захисту.

Згідно з узагальненими результатами, найкращим рішенням серед розглянутих продуктів став Bitdefender Internet Security. Найнижчі результати продемонстрували Zillya! Total Security та Avast Free AiVirus, що свідчить про їх меншу здатність протистояти сучасним загрозам.

Отримані висновки дали змогу об’єктивно визначити сильні й слабкі сторони кожного антивірусу, а також сформувати базу для розробки власного антивірусного засобу, з урахуванням оптимального набору функцій та архітектурних рішень.

2 ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ОСНОВИ РОЗРОБКИ АНТИВІРУСНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Формулювання цілей, функціональне призначення та ключові вимоги до програмного засобу

Розроблений програмний інструмент має функціональне призначення – ефективно протидіяти комп'ютерним вірусам та іншим шкідливим об’єктам, які можуть бути інфільтровані у файлові сховища, програмне забезпечення, документи чи всю операційну систему загалом. Основною метою створення цього засобу є гарантування безпеки комп'ютерних інформаційних систем та збереження недоторканності конфіденційних даних.

У сучасному цифровому середовищі, де ризик втрати інформації через різновиди вірусних загроз є надзвичайно високим], особливої актуальності набуває розуміння, як запобігти цим шахрайським діям. Нагадаємо, що захист інформації охоплює комплекс заходів, спрямованих на запобігання витоку, викраденню, модифікації, розповсюдженню або втраті даних.

Ключові завдання, які покладаються на систему захисту комп'ютерних інформаційних систем, включають:

Виявлення зловмисних компонентів шляхом сканування файлових об'єктів та програм

Забезпечення можливості використання засобу.

Для успішного виконання свого призначення до програмного засобу «AiVirus» висуваються такі основні вимоги та функціональні можливості:

1. Цільове сканування об'єктів (Вибіркове сканування файлів).Користувачеві має бути надана можливість самостійно обирати та ініціювати перевірку будь-якого окремого завантаженого файлу чи застосунку на предмет присутності шкідливого програмного забезпечення.

2. Комплексна системна перевірка (Повне сканування системи).

Повинна бути реалізована функція повноцінної перевірки операційної системи, яка може відбуватися у фоновому (тихому) режимі.

3. Актуалізація сигнатур. Для забезпечення високої ефективності у розпізнаванні нових загроз користувачу необхідно регулярно оновлювати базу ключів (сигнатур).

4. Ізоляція підозрілих об'єктів. Програмний продукт повинен мати спеціально відведене місце для тимчасової ізоляції файлів, програм або документів, які можуть містити потенційну загрозу.

5. Формування антивірусної бази.Створення та підтримка бази даних, де зберігаються вірусні сигнатури. Це дозволить користувачеві самостійно додавати нові ключі, які будуть використані для знищення відповідних шкідливих файлів під час подальшого сканування.

6. Ергономіка та простота користувацького інтерфейсу.Ключовою вимогою є забезпечення легкості у використанні, при якому всі елементи керування та кнопки мають бути максимально доступно розташовані для користувача.

Вимоги до інтерфейсу та обробки даних:

Меню та навігація.Інтуїтивно зрозуміла структура меню сприяє швидкому доступу до всіх основних функцій, зокрема до налаштувань програми, сканування системи та оновлення сигнатурної бази

Режими відображення.Передбачається наявність денного та нічного режимів, що надає користувачеві вибір візуального оформлення.

Гнучкість опцій сканування.Легкодоступні налаштування сканування (наприклад, швидке або повне) дають можливість користувачеві обирати оптимальний режим, виходячи з його потреб та часових ресурсів.

Адаптивність.Ефективний інтерфейс має демонструвати здатність до адаптації при виконанні складних завдань, спрощуючи процес формування запитів та забезпечуючи швидке й легке сприйняття отриманих результатів

Організація обробки інформації. Звітність та обробка інформації про

сканування мають бути зосереджені в одному з кодових файлів програми.Такий підхід запобігає надмірному навантаженню на антивірусний засіб, концентруючи всі дані в єдиному місці. Отримання звітності про сканування фай..

Саме поєднання строгості математичного апарату з можливістю врахування експертного досвіду та інтуїції робить метод аналізу ієрархій особливо цінним для розв’язання задач у сфері розробки та управління IT-проєктами, де часто доводиться порівнювати альтернативи за великою кількістю різнорідних критеріїв.

2.2 Вибір методології розробки програмного продукту

На початковій стадії створення будь-якого програмного забезпечення ключовим є вибір та обґрунтування моделі життєвого циклу (МЖЦ). Обрана методологія суттєво впливає на ефективність планування, якість реалізації, управління ресурсами, ризиками та, зрештою, на успіх усього проєкту. Вона встановлює послідовність робочих етапів, порядок контролю та механізми взаємодії між ними.

Серед існуючого розмаїття підходів до розробки — таких як ітеративні (наприклад, гнучкі методики Agile), лінійні/послідовні (Каскадна або Водоспадна модель) та комбіновані (Спіральна модель) — необхідно визначитися з тим, що оптимально відповідає специфіці програмного засобу «AiVirus».

Проєкт зі створення антивірусного програмного продукту характеризується чітко сформованими вимогами до функціоналу та не передбачає істотних модифікацій у процесі розробки. Враховуючи необхідність послідовного, структурованого та систематичного виконання всіх робіт, для даного проєкту було обрано Каскадну (Водоспадну) модель життєвого циклу (Waterfall Model).

Цей підхід передбачає суворе послідовне проходження всіх фаз розробки, причому перехід до наступної фази можливий виключно після повного завершення та документування результатів попередньої. Основні послідовні етапи цієї моделі включають:

1. Формулювання та аналіз вимог Детальне визначення всіх функціональних і нефункціональних вимог, які висуваються до системи.

2. Проєктування Розробка архітектури, дизайну інтерфейсу та структури внутрішніх програмних модулів.

3. Впровадження (Кодування) Написання програмного коду відповідно до розробленого проєкту.

4. Тестування та налагодження Верифікація працездатності та валідація ПЗ для виявлення та усунення помилок.

5. Експлуатація та супровід Встановлення готового продукту та подальше його обслуговування.

Етапи каскадної моделі життєвого циклу, що наочно демонструють її лінійний характер, представлені на Рисунок 2.1 – Етапи каскадної моделі життєвого циклу.

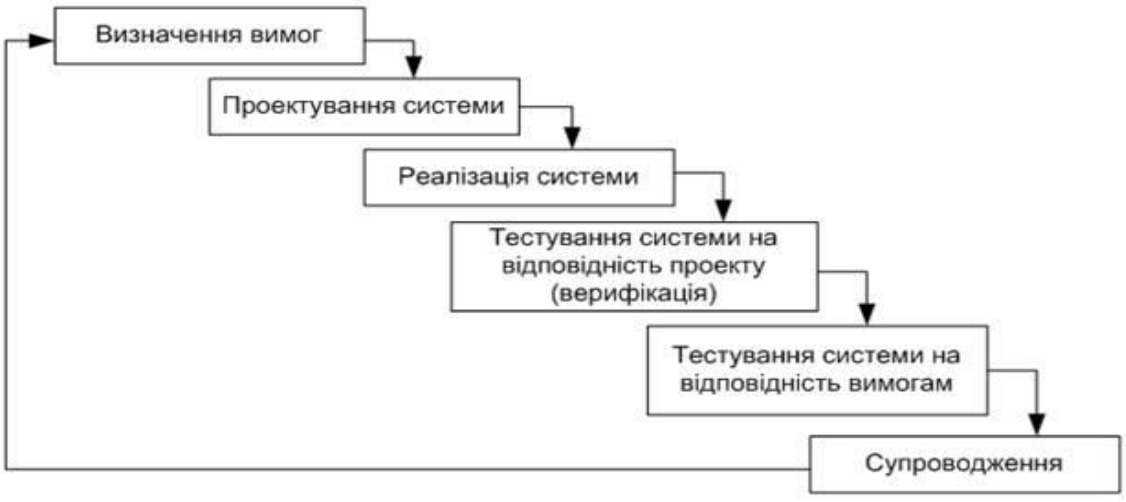

Рисунок 2.1 – Етапи каскадної моделі життєвого циклу

Обґрунтування застосування Каскадної моделі

Вибір цієї методології є виправданим завдяки низці факторів:

Визначеність вимог Вимоги до антивірусного засобу, що були детально описані в попередньому підрозділі, є стабільними та мають мінімальну ймовірність суттєвої зміни протягом виконання проєкту.

Висока керованість Модель гарантує високий рівень організації та контролю на кожному етапі, що є критично важливим для програм, призначених для забезпечення безпеки, де будь-яка помилка може мати критичні наслідки.

Чітка документація Детальне документування результатів кожної фази значно полегшує розуміння статусу проєкту та передачу інформації.

Таким чином, для реалізації програмного засобу «AiVirus» була обрана Каскадна модель, оскільки вона найкращим чином забезпечує послідовний, структурований і добре контрольований процес створення продукту з фіксованим набором функціональних вимог.

2.3 Обґрунтування вибору інструментів та технологій для розробки програмного засобу

Після визначення цілей, функціональних вимог та моделі життєвого циклу важливим етапом є вибір технологічних засобів, що забезпечать стабільність, гнучкість і ефективність реалізації антивірусного програмного продукту. Вибір інструментів має базуватися на здатності обраної технології обробляти великі обсяги файлових даних, забезпечувати швидкі операції сканування та підтримувати взаємодію з файловою системою й мережевими ресурсами для оновлення сигнатур.

Основними критеріями вибору середовища розробки та технологічного стеку стали:

стабільність і надійність обробки даних при роботі з файлами різних типів;

доступність бібліотек, необхідних для кодування, декодування,

мережевих запитів і побудови інтерфейсу;

легкість підтримки та модифікації проєкту;

кросплатформеність, що дозволяє розширити програму у майбутньому;

можливість швидкої інтеграції з діючими методами сканування, аналізу та карантинування вірусів.

З огляду на ці вимоги, для розробки «AiVirus» було обрано:

1. Мову програмування Python

Python забезпечує оптимальний баланс між простотою синтаксису та широкими можливостями обробки даних. Його популярність у сфері кібербезпеки пояснюється великою кількістю спеціалізованих бібліотек, що дозволяють реалізувати:

роботу з файловою системою (os, pathlib);

мережеві запити (urllib.request);

багатопоточність (threading);

кодування та декодування даних (base64);

модульність системи та швидке тестування функціональних блоків.

Високий рівень читабельності коду також сприяє подальшому супроводу, розширенню та тестуванню програмного засобу.

Розробницьке середовище VS Code

Visual Studio Code обрано як основне середовище розробки завдяки таким можливостям:

широкій підтримці Python;

зручним засобам відлагодження (debugging);

модульній структурі з під’єднанням необхідних розширень;

низьким вимогам до ресурсів системи;

підтримці Git для контролю версій.

VS Code дає змогу працювати з проєктом у структурованому вигляді, забезпечує добру інтеграцію з терміналом, віртуальними середовищами Python та сторонніми бібліотеками.

3. Бібліотека Tkinter для побудови GUI

Оскільки одним із ключових вимог є простота та інтуїтивність інтерфейсу, для створення графічного середовища взаємодії з користувачем було застосовано Tkinter. Цей вибір забезпечує:

мінімальне навантаження на систему;

зручну реалізацію кнопок, вікон, панелей та повідомлень;

можливість налаштування кольорових режимів (денний/нічний);

просту інтеграцію з функціоналом програми.

Tkinter добре підходить для легких антивірусних інструментів, які не потребують складної графіки, але вимагають зрозумілої взаємодії.

4. Зовнішні інтернет-ресурси для оновлення сигнатур

Оновлення антивірусної бази базується на можливості отримання оновлених сигнатур через Інтернет. У проєкті використовується підхід завантаження даних із веб-ресурсів через:

urllib.request` — для запитів до сторінок зі списками вірусних ключів;

обробку HTML-вмісту для пошуку актуальних посилань;

формування та зберігання локальної бази у вигляді текстових файлів.

Ця функція забезпечує можливість регулярного оновлення без участі користувача, що підвищує ефективність виявлення нових загроз.

5. Формування карантину на основі Base64-кодування

Для тимчасової ізоляції підозрілих файлів використано механізм

кодування у формат Base64, що дозволяє:

перетворювати вміст файла у безпечний текстовий формат;

гарантувати неможливість випадкового запуску шкідливого коду;

зберігати файли у карантині без ризику зараження системи;

легко виконувати зворотне декодування при відновленні.

Такий підхід є практичним рішенням для легких антивірусів та відповідає вимогам безпеки.

6. Підтримка багатопоточності

Оскільки повне сканування системи може тривати значний час, використання багатопоточності дає можливість:

розподіляти навантаження на декілька потоків,

прискорювати обробку великої кількості файлів,

забезпечувати кращу реакцію інтерфейсу під час сканування.

Багатопоточність реалізована через модуль threading.

Загальне обґрунтування вибору технологій

Застосування Python, tkinter, Base64-кодування, багатопоточності та мережевих запитів створює комплексну платформу для побудови ефективного антивірусу легкого рівня. Обраний технологічний стек повністю задовольняє:

функціональні вимоги до інструмента;

потребу в швидкості та зручності тестування;

необхідність у створенні доступного графічного інтерфейсу;

можливість масштабування та подальшого розвитку проєкту.

Таким чином, вибрані технології та інструменти становлять раціональний і технічно обґрунтований фундамент для реалізації програмного засобу «AiVirus».

3 РОЗРОБКА, РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Опис проєкту

Програмний засіб «AiVirus» розроблено як інструмент для посилення рівня захисту комп'ютерних інформаційних систем від загроз, спричинених шкідливим програмним забезпеченням, таким як віруси, трояни та інші зловмисні програми.

3.1.1. Архітектурна концепція

Розробка програмного засобу була виконана на основі трирівневої архітектурної моделі , яка забезпечує чітке розділення функціональних обов'язків. Ця архітектура дозволяє реалізувати систему з високою гнучкістю, простотою модифікації та легкістю підтримки. Вона складається з таких ключових компонентів:

1 Рівень користувацького інтерфейсу (Presentation Tier) . Цей рівень є точкою взаємодії програми та користувача. Його основна функція — відображення інформації (результатів сканування, статусних повідомлень) та отримання команд від користувача (запуск перевірки, оновлення баз, вибір об'єктів для сканування). Він відповідає за візуальну частину програми.

2 Рівень бізнес-логіки (Application/Business Logic Tier) . Це центральна частина системи, що містить основні алгоритми та функціонал антивірусу. На цьому рівні відбувається вся ключова обробка даних: запуск процесу сканування файлів, порівняння їхніх сигнатур із наявною базою загроз, прийняття рішень щодо заражених об'єктів (видалення, лікування,

переміщення до карантину).

3 Рівень даних (Data Tier) . Цей компонент забезпечує постійне зберігання інформації, необхідної для роботи програми. Сюди належить база вірусних сигнатур (ключів), а також лог-файли та журнали, які фіксують результати сканування та історію дій програми.

Графічне представлення архітектури програмного продукту, що відображає взаємодію вказаних рівнів, наведено на Рисунок 3.1 – Архітектура програмного засобу .

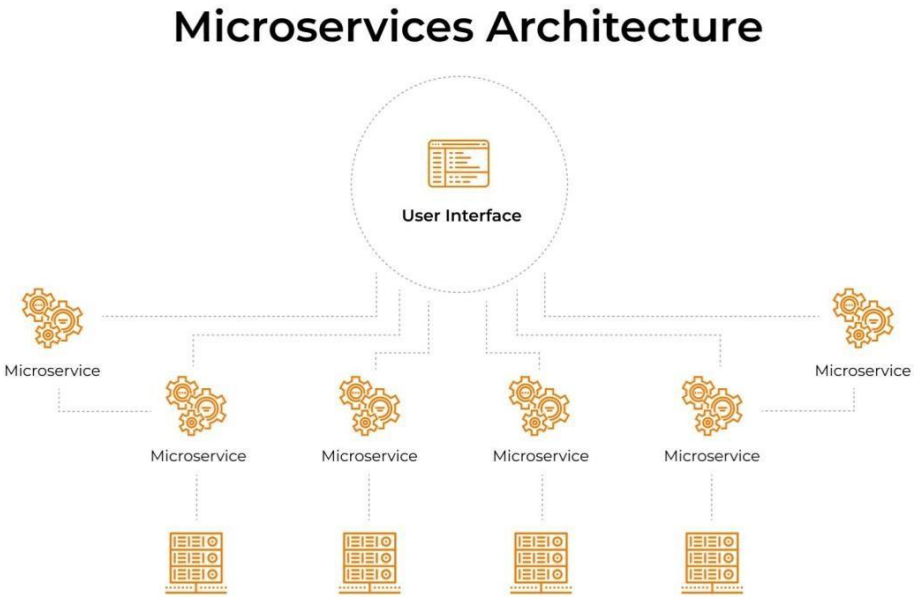

Рисунок 3.1 – Архітектура програмного засобу .

3.1.2. Принцип функціонування

Робота антивірусного засобу «AiVirus» ґрунтується на сигнатурному методі виявлення загроз. Це означає, що програма порівнює вміст файлів, які перевіряються, з шаблонами (сигнатурами) відомих шкідливих об'єктів, що зберігаються в антивірусній базі.

Алгоритм роботи має таку послідовність:

1 Ініціалізація сканування. Користувач запускає перевірку, обираючи

об’єкт (окремий файл, каталог, або повне сканування системи).

2 Формування хешу та порівняння сигнатур. У режимі реального часу програма зчитує та обробляє обраний об'єкт. Кожен файл генерує унікальний ідентифікатор (хеш), який потім зіставляється з хешами, записаними в локальній базі сигнатур.

3 Виявлення збігів. Якщо знайдено повний або частковий збіг, файл ідентифікується як потенційно небезпечний або заражений .

4 Виконання дій. Після ідентифікації загрози програма пропонує користувачеві низку дій:

Лікування: Спроба видалити вірусний код, зберігаючи при цьому «здорову» частину файлу.

Видалення: Безповоротне знищення зараженого файлу.

Карантин: Переміщення підозрілого файлу до ізольованого сховища, де він не може завдати шкоди системі.

5 Формування звіту. Результати перевірки, включаючи інформацію про виявлені загрози та виконані дії, фіксуються та відображаються користувачеві.

Схематичне зображення внутрішньої структури та основних функціональних елементів програми, що деталізує механізм взаємодії модулів, можна побачити на Рисунок 3.2 – Схема функціональних можливостей програмного засобу «AiVirus» .

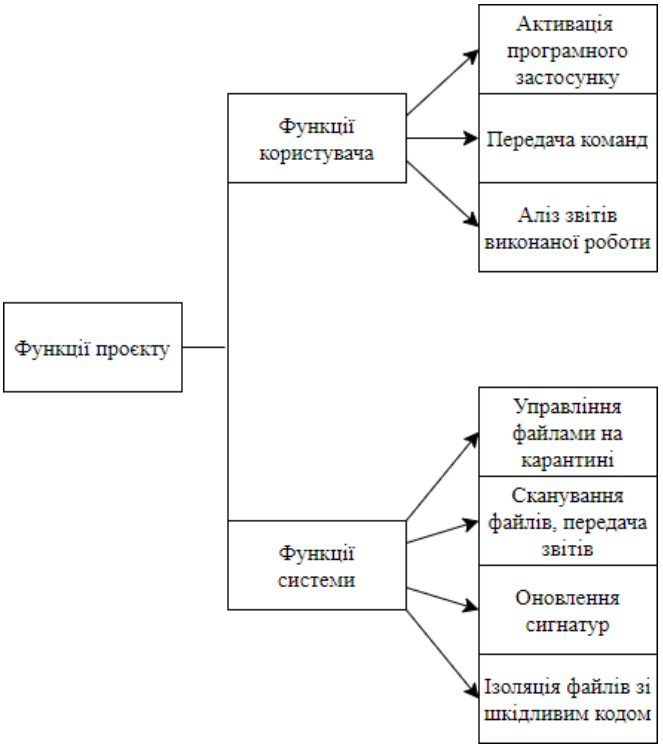

Рисунок 3.2 – Схема функціональних можливостей програмного засобу

Таким чином, розроблений проєкт «AiVirus» являє собою структуровану, модульну програму, здатну ефективно ідентифікувати та нейтралізувати загрози завдяки використанню актуальної бази вірусних сигнатур.

3.2 Обґрунтування вибору технологічного стеку та інструментальних засобів для розробки

Вибір відповідного набору технологій та інструментальних засобів є вирішальним етапом, що безпосередньо впливає на швидкість, ефективність, надійність та супровідність кінцевого програмного продукту. Для реалізації проєкту «AiVirus» були обрані засоби, які найкращим чином відповідають поставленим функціональним вимогам та архітектурній концепції.

3.2.1. Вибір мови програмування

В якості основної мови для розробки було обрано Python. Цей вибір обґрунтований наступними ключовими перевагами:

Простота та швидкість розробки: Python є високошвидкісною мовою для написання коду завдяки її лаконічному та чистому синтаксису, що значно прискорює процес реалізації.

Велика кількість бібліотек: Наявність потужної екосистеми з готовими бібліотеками (наприклад, для роботи з файловою системою, обробки даних, хешування) дозволяє уникнути написання базового функціоналу з нуля.

Кросплатформність: Python забезпечує можливість запуску розробленого програмного засобу на різних операційних системах без значних змін у коді.

Актуальність: Широке використання мови у сфері кібербезпеки та обробки великих обсягів даних є додатковим аргументом на її користь.

3.2.2. Вибір середовища розробки

Для роботи над кодом та управління проєктом було обрано інтегроване середовище розробки Visual Studio Code (VS Code). Це рішення підкріплене такими перевагами:

Легкість та швидкодія: VS Code є відносно "легким" редактором, що забезпечує швидкий запуск та високу продуктивність навіть на системах з обмеженими ресурсами.

Гнучкість та розширюваність: Завдяки системі розширень (Extensions) VS Code легко адаптується під потреби проєкту, надаючи інтегровані можливості для налагодження (debugging), роботи з Git та підтримки специфічного синтаксису Python.

Інтеграція з терміналом: Вбудований термінал спрощує виконання

команд, пов'язаних з тестуванням та компіляцією, без необхідності перемикання між різними вікнами.

Зручність інтерфейсу: Інтерфейс VS Code є інтуїтивно зрозумілим, що сприяє високій концентрації розробника на процесі написання коду.

3.2.3. Додаткові інструменти та бібліотеки

Для забезпечення необхідного функціоналу «AiVirus» використовувалися наступні критично важливі бібліотеки та компоненти Python:

Операційна система (OS) та файлова система (Pathlib): Використання вбудованих модулів Python для ефективної взаємодії з файловою системою, навігації по каталогах та перевірки властивостей файлів під час сканування.

Хешування (Hashlib): Бібліотека hashlib є незамінною для реалізації сигнатурного методу, оскільки вона дозволяє генерувати унікальні хеші (відбитки) для файлів. Ці хеші потім порівнюються з ключами у вірусній базі для ідентифікації загроз.

Створення графічного інтерфейсу (GUI): Для розробки користувацького інтерфейсу, що відповідає вимогам ергономіки, була обрана бібліотека Tkinter (або аналогічна, як-от PyQt/Kivy). Це дозволяє створити інтуїтивно зрозуміле та візуально привабливе вікно програми для взаємодії з користувачем.

Таким чином, комбінація Python як потужної мови, VS Code як гнучкого середовища розробки та спеціалізованих бібліотек створює оптимальну інструментальну базу для успішної та якісної реалізації проєкту «AiVirus».

3.3 Програмування програмного засобу

Розроблення програмного засобу «AiVirus» передбачало створення функціональних модулів, здатних забезпечити виявлення шкідливих файлів, обробку результатів аналізу та інтерактивну взаємодію з користувачем. Основна увага під час програмування приділялася реалізації алгоритмів сканування, формуванню сигнатурної бази, а також забезпеченню стабільної роботи системи в умовах обробки великих обсягів даних.

На початковому етапі проєктування необхідно було визначити мову програмування та бібліотеки, що дозволяють реалізувати всі вимоги до майбутнього продукту — від сканування файлів до роботи з базами сигнатур і реалізації графічного інтерфейсу. Архітектура застосунку включає шість ключових функціональних компонентів: вибіркове сканування, повне сканування операційної системи, блок оновлення сигнатур, модуль карантину, сигнатурну базу та інтерфейс користувача.

Основні програмні компоненти системи

1. Вибіркове сканування файлів.

Модуль призначений для перевірки окремих файлів або програм, обраних користувачем. Після завантаження файлу система обчислює його MD5-хеш, порівнює отримане значення з наявними сигнатурами та за результатом визначає, чи містить файл загрозу.

2. Повне сканування системи.

Передбачає перевірку всіх доступних каталогів та підкаталогів. У режимі повного сканування модуль працює у «тихому» режимі, тобто дані опрацьовуються у фоновому режимі, а результати записуються до спеціального файлу звіту, що дозволяє уникнути перевантаження інтерфейсу повідомленнями.

3. Оновлення сигнатурної бази.

Система здійснює перевірку актуальності списків сигнатур, завантажуючи нові ключі з доступних джерел. Модуль аналізує вже завантажені посилання, порівнює їх зі списком актуальних, після чого додає нові дані до файлів сигнатур. Завдяки цьому користувач може підтримувати базу у актуальному стані.

4. Карантин.

Компонент ізолює підозрілі або заражені файли, перетворюючи їх у безпечний формат (наприклад, шляхом кодування base64). У карантині доступні такі функції, як видалення окремого файлу, повне очищення, відновлення вибраного об'єкта або перенесення всіх файлів назад у систему.

5. Сигнатурна база.

Це сукупність хешів та сигнатур вірусів, з якими порівнюються файли під час сканування. База є попередньо заповненою та може містити понад 500 000 ключів. Користувач має можливість власноруч додавати знайдені значення, що дозволяє адаптувати систему під конкретні потреби.

6. Графічний інтерфейс.

Інтерфейс розроблений таким чином, щоб усі основні елементи керування були доступними без додаткових переходів. Прості та інтуїтивно зрозумілі кнопки дозволяють швидко запускати вибіркове та повне сканування, оновлювати базу або працювати з карантином.

Алгоритм роботи антивіруса

Алгоритм сканування побудовано таким чином, щоб забезпечити послідовне опрацювання всіх розділів операційної системи та мінімізувати навантаження на ресурси комп’ютера. Загальний процес можна поділити на кілька основних етапів.

1. Визначення розділів системи

Для ОС Windows програма переглядає можливі дискові літери (A–Z) та формує список наявних розділів. Якщо система не є Windows, програма одразу переходить до формування індексів, минаючи перевірку дисків.

2. Отримання списку каталогів та файлів

За допомогою бібліотеки `glob` здійснюється збір інформації про каталоги та файли. Для Windows використовується формат шляху з подвійним бекслешем («\»), для UNIX-подібних систем — слеш «/».

Отримані каталоги записуються у змінну `verzeichnisse`.

3. Фільтрація та підготовка до сканування

Система відокремлює каталоги від файлів, після чого формує список файлів (`files`), що підлягають аналізу.

4. Збереження списку файлів

Системою формується спеціальний файл зі списком шляхів, що надалі використовується під час глибокого сканування.

5. Обчислення MD5-хешів та пошук загроз

Під час вибіркового сканування користувач обирає файл, після чого програма:

генерує MD5-хеш файлу,

порівнює отриманий хеш із сигнатурами,

у разі збігу — переміщує файл до карантину та кодує його в безпечному форматі.

Результат виводиться у вигляді текстового звіту, де зазначаються час виконання та наявність загроз.

6. Оновлення бази сигнатур

Під час оновлення програма:

зчитує файли `links_current` та `links_downloaded`,

визначає нові посилання,

авантажує дані за зазначеними URL,

додає нові сигнатури до основного файлу.

У випадку відсутності нових оновлень користувач отримує відповідне повідомлення.

Першим етапом розроблення є підключення необхідних бібліотек. Саме

з цього починається робота сценарію, оскільки модулі забезпечують доступ до інструментів, потрібних для функціонування застосунку. Зокрема, за допомогою бібліотеки tkinter формується графічний інтерфейс користувача, що дає можливість реалізувати зручну та інтуїтивну взаємодію з антивірусною програмою.

Як показано на рисунку 3.3, здійснюється створення каталогів для операційної системи Windows. На цьому етапі формується структура папок і службових файлів, які використовуються для зберігання різних компонентів антивірусного програмного забезпечення.

```
28 if "win" in os_name:
29     if not os.path.exists("AntiVirus\\Quarantine\\"):
30         os.makedirs("AntiVirus\\Quarantine\\")
31     if not os.path.exists("AntiVirus\\sf\\"):
32         os.makedirs("AntiVirus\\sf\\")
33     if not os.path.exists("AntiVirus\\Large_Update_File\\"):
34         os.makedirs("AntiVirus\\Large_Update_File")
35     quarantine_folder = "AntiVirus\\Quarantine\\*"
36     file_to_quarantine = "AntiVirus\\Quarantine\\"
37     partitionen_folder = "AntiVirus\\sf\\sf.txt"
38     links_current = "AntiVirus\\Large_Update_File\\links_current.txt"
39     links_downloaded = "AntiVirus\\Large_Update_File\\links_downloaded.txt"
40     large_signatures = "AntiVirus\\Large_Update_File\\signatures.txt"
41     f = open(partitionen_folder, "a")
42     f.close()
43     f = open(links_current, "a")
44     f.close()
45     f = open(links_downloaded, "a")
46     f.close()
47     f = open(large_signatures, "a")
48     f.close()
```

Рисунок 3.3 – Створення каталогів для ОС Windows

Аналогічну структуру каталогів було сформовано і для інших операційних систем, окрім Windows. Після завершення підготовки файлового середовища розпочалася робота над графічним інтерфейсом користувача (GUI). На цьому етапі були створені основні елементи керування: кнопка оновлення update_button), кнопка вибіркового сканування scan_button), повного сканування fullscan_button) та кнопка завершення роботи програми

quit_button).

Окрему увагу приділено модулю карантину, який було доповнено низкою функціональних кнопок: видалення окремого файлу b_delete), повне очищення карантину b_delete_all), відновлення вибраного елемента b_restore) та відновлення всіх ізольованих файлів b_restore_all). Також реалізовано кнопку додавання файлу b_add_file) для ручного поповнення списку.

На завершальному етапі оформлення інтерфейсу були проведені додаткові налаштування, зокрема коригування яскравості відповідно до поточного часу доби. На рисунку 3.4 зображено процес конфігурації цих параметрів.

```
if daytime >= 18 or daytime <= 4:
    bgc = "black"
    fgc = "white"
    special = "brown"
    special_text = "(°_°)⊃ ☆ Доброго дня " + os.getlogin() + " ☆ ⊂(°_°)\n"
elif daytime > 4 and daytime <= 8:
    special_text = "\(o¯∀¯o)/ Доброго вечора " + os.getlogin() + " \(o¯∀¯o)/\n"
    bgc = "#b4d60c"
    fgc = "black"
    special = "orange"
else:
    bgc = "white"
    fgc = "black"
    special = "#1ccaed"
    special_text = "\(≧∇≦)/ Ласкаво просимо " + os.getlogin() + " \(≧∇≦)/\n"

```

Рисунок 3.4 – процес конфігурації цих параметрів

1. Створення системи сканування

На рисунку 3.5 наведено код, що реалізує процес сканування файлів, їх фільтрацію та подальше збереження результатів. Даний фрагмент відповідає за визначення доступних накопичувачів, перегляд їхніх каталогів та збір інформації про вміст дисків. Алгоритм починається з отримання переліку розділів, після чого програма переходить до аналізу папок і файлів на кожному з них. Додатково в цьому блоці визначаються параметри часу доби,

що використовуються для внутрішньої логіки роботи застосунку.

Функція partitions формує список наявних у системі дисків та зберігає його у структурі `partitionen`. Після цього викликається функція indeces, яка здійснює подальше глибинне сканування.

У межах indeces для кожної знайденої партіції виконується рекурсивний обхід усіх вкладених папок, що дозволяє отримати повну вибірку каталогів і файлів без пропусків.

Після завершення обходу дані проходять етап фільтрації:

усі знайдені каталоги заносяться до списку verzeichnisse,

усі доступні файли — до списку files.

Згодом інформація щодо файлів зберігається в окремому службовому файлі, який надалі використовується під час процесу перевірки.

На рисунку 3.6 представлено механізм розподілу навантаження між потоками. У коді здійснюється визначення часток роботи: перший потік отримує 12,5% від загальної кількості файлів, другий — 25%, і так далі. Таким чином кожен потік обробляє свій окремий сегмент файлового списку, що підвищує швидкість і ефективність загального процесу сканування.

```
if "win" in os_name:
    for ind in range(len(partitionen)):
        #print(partitionen[ind])
        while verzeichnisse2 != []:
            verzeichnisse2 = glob.glob(partitionen[ind] + "\\*"*x)
            for i in range(len(verzeichnisse2)):
                verzeichnisse.append(verzeichnisse2[i])
            x += 1
        x = 1

    for i in range(len(verzeichnisse)):
        if "." in verzeichnisse[i]:
            files.append(verzeichnisse[i])
    for i in range(len(verzeichnisse)):
        if not os.path.isfile(verzeichnisse[i]):
            verzeichnisse_tmp.append(verzeichnisse[i])
    verzeichnisse = verzeichnisse_tmp
    i = 0
    f = open(sfsFolder, "w")
    for i in range(len(files)):
        f.write(files[i] + "\n")
    f.close()
    time.sleep(3)
```

Рисунок 3.5 – Сканування файлів, фільтрація та збереження

```
if part == 1:#Thread-1
    i = int(len(files)*0.125)
    tmp = 0
if part == 2:#Thread-2
    i = int(len(files)*0.25)
    tmp = int(len(files)*0.125)
if part == 3:#Thread-3
    i = int(len(files)*0.375)
    tmp = int(len(files)*0.25)
if part == 4:#Thread-4
    i = int(len(files)*0.5)
    tmp = int(len(files)*0.375)
if part == 5:#Thread-5
    i = int(len(files)*0.625)
    tmp = int(len(files)*0.5)
if part == 6:#Thread-6
    i = int(len(files)*0.75)
    tmp = int(len(files)*0.625)
if part == 7:#Thread-7
    i = int(len(files)*0.875)
    tmp = int(len(files)*0.75)
if part == 8:#Thread-8
    i = int(len(files))
    tmp = int(len(files)*0.875)
```

Рисунок 3.6 – Розподіл роботи між потоками

Після завершення процесу сканування система формує підсумковий звіт, у якому зазначаються основні параметри перевірки: тривалість сканування,

кількість виявлених загроз, поточний стан системи, а також повідомлення про її безпечність або наявність потенційних ризиків.

2. Створення тимчасового сховища — карантину

На рисунку 3.7.наведено фрагмент коду, що відповідає за створення механізму ізоляції файлів — карантину. У цьому модулі реалізовані функції кодування та декодування файлів у форматі Base64, що дозволяє безпечно зберігати підозрілі об’єкти в ізольованому середовищі та унеможливлює їх випадкове виконання чи поширення.

```
def encode_base64(file, qPath):
    global os_name

    org_file_path = bytes(file, "utf-8")
    if "win" in os_name:
        org_file_name = file.rfind("\\")
    else:
        org_file_name = file.rfind("/")
    org_file_name = file[org_file_name+1:]
    f = open(file, "rb")
    org_content = f.read()
    f.close()
    os.remove(file)
    new_content = base64.b64encode(org_content)
    f = open(qPath + org_file_name + ".eb64", "wb")
    f.write(org_file_path + b"\n")
    f.write(new_content)
    f.close()
```

Рисунок 3.7 – Кодування файлів в формат Base64 та їх подальше збереження їх в карантині.

Після виконання функції `def encode_base64(file, qPath):` програма отримує байтове представлення шляху до вихідного файлу. Далі визначається назва файлу з урахуванням особливостей операційної системи, після чого відбувається відкриття оригінального файлу для читання у бінарному режимі.

Команда `org_content` здійснює зчитування вмісту файлу, після чого вихідний файл видаляється з файлової системи, щоб уникнути можливості його подальшого використання у зараженому вигляді.

Операція

`new_content = base64.b64encode(org_content)`виконує перетворення прочитаних даних у формат Base64.

Закодований вміст записується в новий файл за допомогою таких команд:

- f = open(qPath + org_file_name + «.eb64», «wb»)
- f.write(org_file_path + b»\n»)
- f.write(new_content)
- f.close()

Таким чином створюється безпечна копія файлу, що зберігається у карантині.

Функція для зворотного процесу — декодування файлів, попередньо переведених у формат Base64, починається з оголошення

`def decode_base64(file):`, як показано на рисунку 3.8, де представлено приклад реалізації процедури відновлення файлів.

```python
def decode_base64(file):
    f = open(file, "rb")
    org_content = f.read()
    f.close()
    org_content = org_content.splitlines()
    org_file_path = org_content[0]
    org_content.remove(org_file_path)
    new_content = []
    for i in org_content:
        new_content.append(base64.b64decode(i))
    f = open(org_file_path, "wb")
    for i in new_content:
        f.write(i + b"\n")
    f.close()
    os.remove(file)
```

Рисунок 3.8 – Декодування файлів

Процес декодування відрізняється від кодування тим, що спочатку зчитує весь вміст файлу та розбиває його на окремі рядки за допомогою змінної `org_content`. Далі створюється порожній список, призначений для збереження відновлених даних. Кожний рядок, що містить закодовану інформацію, проходить декодування з формату Base64 та додається до нового списку. Після формування повного набору даних декодований вміст записується у відновлений оригінальний файл. Завершальним кроком є видалення закодованої копії з карантину.

Якщо описати алгоритм більш узагальнено, то для кодування файлу необхідно:

визначити шлях до вихідного файлу, який потрібно ізолювати;

відокремити його ім’я від повного шляху;

зчитати вміст та видалити оригінальний файл із файлової системи;

перетворити зчитані дані у формат Base64;

зберегти шлях до оригіналу та закодований вміст у новий файл, що розміщується в каталозі карантину.

На цьому етап кодування вважається завершеним.

Процес декодування виглядає так:

програма зчитує закодований файл повністю;

розділяє його на текстові рядки;

вилучає перший рядок, який містив шлях до вихідного файлу;

кожен наступний рядок декодується з Base64 і додається до списку даних;

отриманий вміст записується у відновлений файл за його початковим шляхом;

після успішного завершення операції закодований файл видаляється з каталогу карантину, щоб уникнути дублювання.

Після реалізації всієї логіки роботи карантину можна переходити до подальшого оформлення інтерфейсу програми. На рисунку 3.9 зображено вікно інтерфейсу, яке відображає результати роботи карантинного модуля.

```
terminations = glob.glob(quarantine_folder)
if terminations == []:
    text_box.insert(END, "[ + ] Нема файлів в карантині\n", "positive")
    text_box.tag_config('positive', foreground="green")
    text_box.see(END)
    text_box.update()
else:
    text_box.insert(END, "[ + ] Файли в карантині:\n", "positive")
    text_box.tag_config('positive', foreground="green")
    text_box.see(END)
    text_box.update()
    for i in terminations:
        text_box.insert(END, "[ * ] " + i + "\n", "info")
        text_box.tag_config("info", background = "red")
        text_box.see(END)
        text_box.update()
        li.insert(END, i)
        li.update()
```

Рисунок 3.9 – Результати роботи карантинного модуля

На рис. 3.10 показано код системи вивявлення шкідливого ПЗ в усій операційній системі. Також може відбуватись пошук файлів на наявність вірусних сигнатур.

Базу сигнатур було взято із сховища VirusShare.com. Сайт надає доступ до шкідливого програмного забезпечення. Для того, аби щоразу поновлювати

наявну базу, було створено такий код як на рис. 3.11.

```
            f = open(files[i], "rb")
            file_content = f.read()
            f.close()
        except:
            continue
        ret = scan_auto(files[i])
        if ret == True:
            text_box.insert(END, "[ ! ] Програма: " + files[i] + " може бути шкідливою\n", "important")
            text_box.tag_config("important", foreground="red")
            text_box.see(END)
            text_box.update()
            quarantaene.encode_base64(files[i])
        files_len -= 1
        i -= 1
    runtime = int(time.time() - start)
    text_box.insert(END, "[ + ] Сканування закінчилося після\n " + str(runtime/60) + " хвилин.\n", "positive")
    text_box.tag_config("positive", foreground="green")
    if files_len == 0:
        full_scan["state"] = "normal"
    if len(terminations) == 0:
        text_box.insert(END, "[ +++ ] Ваш ПК безпечний" + "\n", 'important')
    else:
        text_box.insert(END, "[ !!! ] Знайдено {0} загроз на вашому ПК\n".format(len(terminations)))
    text_box.tag_config("important", background="red")
    text_box.see(END)
    text_box.update()
```

Рисунок 3.10 – Код сканування ПК

```
def link_collector(): #gets Links to refresh update-site;short spider
    global text_box
    u_list = []

    text_box.insert(END, "[ * ] Пошук оновлень бази даних...\n")
    text_box.see(END)
    text_box.update()
    u = urllib.request.urlopen("http://virusshare.com/hashes").read().decode("utf-8").splitlines()
    f = open(links_current, "w")
    for i in u:
        if "href='" in i:
            first = i.find("href='") + len("href='")
            i = i[first:]
            last = i.find("'")
            i = i[:last]
        if 'href="' in i:
            first = i.find('href="') + len('href="')
            i = i[first:]
            last = i.find('"')
            i = i[:last]
        if "VirusShare" in i:
            f.write("http://virusshare.com/" + i + "\n")
    f.close()
    return update()
```

Рисунок 3.11 – Завантаження оновлень бази сигнатур

Під час виконання цього фрагмента коду відбувається звернення до веб-ресурсу, що містить актуальну базу даних. За допомогою бібліотеки urllib.request програма отримує HTML-вміст сторінки. Далі отриманий текст

розбивається на окремі рядки методом `splitlines()`, у результаті чого формується список u.

Після цього відкривається файл links_current у режимі запису (`w`). Це дає змогу очистити його попередній вміст і підготувати файл до занесення нових URL-адрес, оскільки після попередніх оновлень у ньому могли залишатися застарілі дані.

Під час проходження кожного рядка зі списку u програма перевіряє, чи містить він фрагмент `href=`. Якщо така підстрока виявлена, код визначає позиції початку та завершення гіперпосилання, після чого вирізає знайдену адресу. У випадку, коли витягнуте посилання містить ключове слово «VirusShare», воно записується до файлу links_current як актуальний елемент бази.

Після обробки всіх рядків викликається функція update(), яка запускає процедуру завантаження нових файлів та оновлення сигнатурної бази.

На рисунку 3.12 наведено фрагмент коду, що реалізує механізм оновлення бази шляхом отримання та збереження нових даних.

```
zaehler = 0
f = open(links_current, "r")
f2 = open(links_downloaded, "r")
files_downloaded = f2.read()
f2.close()
f2 = open(links_downloaded, "r")
for i in f.read().splitlines():
    f2 = open(links_downloaded, "r")
    con = f2.read()
    f2.close()
    f2 = open(links_downloaded, "a")
    if i not in con:
        zaehler += 1
        f2.write(i + "\n")
        f2.close()
        text_box.insert(END, "[ * ] Завантажено:\n"+i)
        text_box.see(END)
        text_box.update()
        signatures = open(large_signatures, "a")
        url = i
        tmp = urllib.request.urlopen(url).read().decode("utf-8").splitlines()

        for j in tmp:
            if j[0] != '#':
                signatures.write(j + "\n")
        signatures.close()
if zaehler == 0:
    text_box.insert(END, "[ * ] Нових оновлень не знайдено\n")
    text_box.see(END)
    text_box.update()
```

Рисунок 3.12 – Код оновлення бази даних сигнатур

3.4 Режим роботи програмного засобу

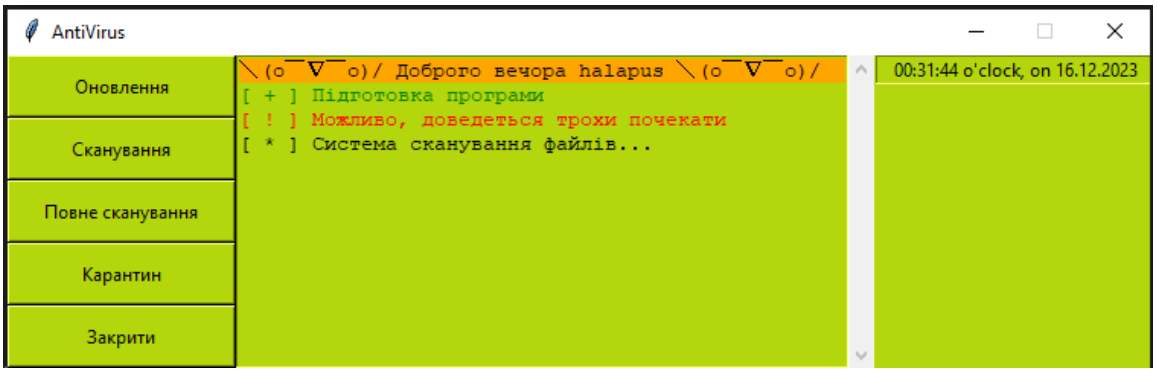

Рисунок 3.13 – Інтерфейс який з’являється після 18:00

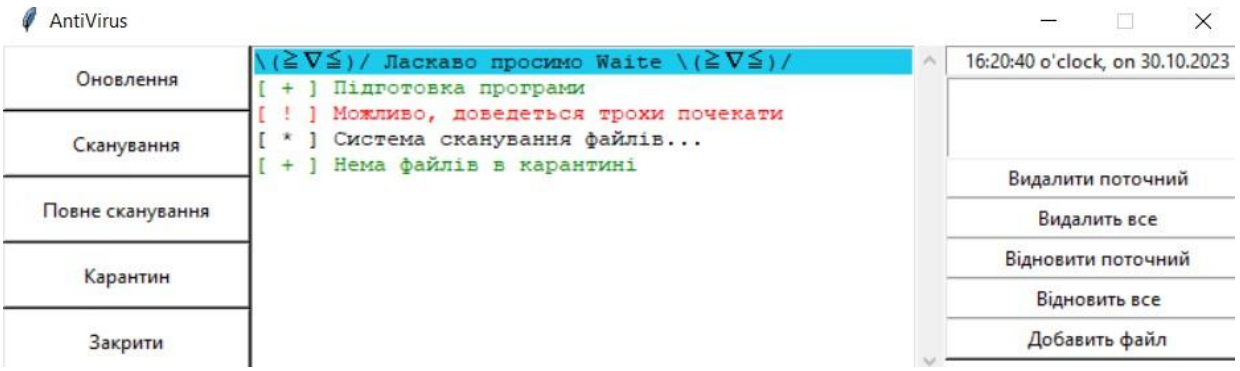

Рисунок 3.14 – Інтерфейс який з’являється з 7:00 до 18:00

Кнопка «Сканування» запускає ручну перевірку вибраного користувачем файлу або програми. Для демонстрації було виконано сканування тестового документа. Після завершення перевірки система відразу відображає тривалість процесу та повідомляє, чи були виявлені загрози.

Розділ «Карантин» у даному прикладі одразу повертає повідомлення про відсутність ізольованих файлів. На правій панелі інтерфейсу можна побачити всі доступні функції інструмента, як показано на рисунку 3.15, де наведено огляд

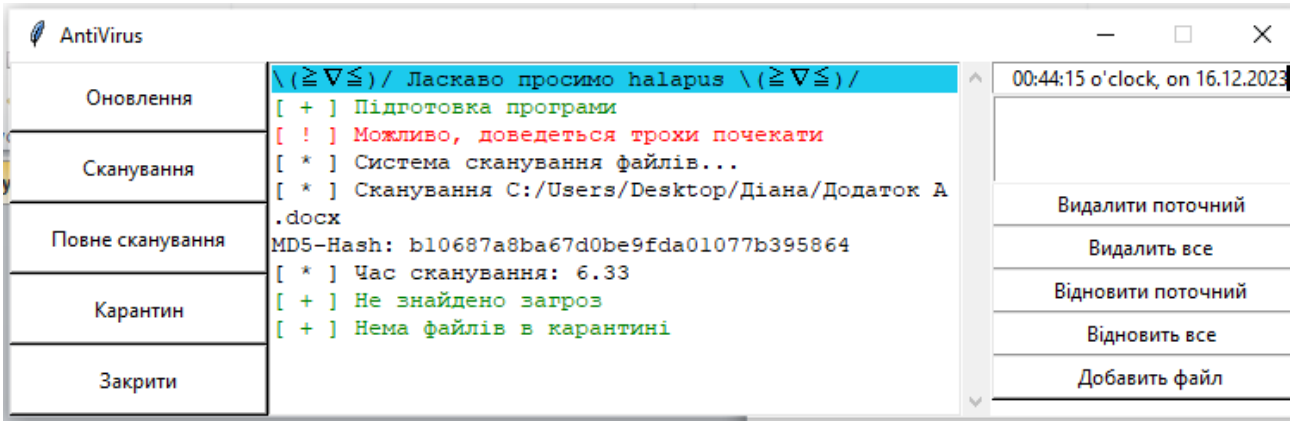

Рисунок 3.15 – Огляд розробки

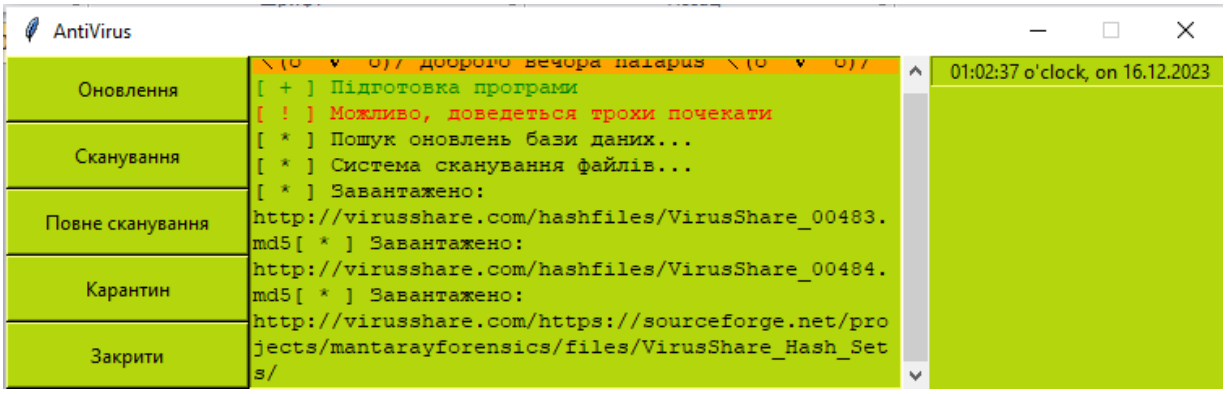

Рисунок 3.16 – Робота системи оновлення

3.5 Організація тестування та налагоджування програмного забезпечення

Тестування програмного забезпечення є ключовим етапом розробки, метою якого є перевірка коректності функціонування створеного продукту та оцінка його відповідності встановленим вимогам. Завдяки тестуванню виявляються помилки, визначаються недоліки у логіці роботи, перевіряється надійність і стабільність функціональних модулів, а також формується впевненість у якості кінцевого результату.

Основним завданням тестування є підтвердження того, що програмний продукт повністю відповідає попередньо визначеним вимогам. Додатково тестування дозволяє оцінити:

поведінку програми під час виконання різних функцій;

відповідність архітектури та інтерфейсу поставленим вимогам;

ступінь реалізації функціональних можливостей, визначених на етапі проєктування;

якість спостереження та оперативність моніторингу під час виконання окремих процесів;

необхідність подальшої оптимізації коду та вдосконалення інтерфейсу.

Для досягнення якісного результату під час тестування важливо порівнювати фактичні результати виконання програми з очікуваними значеннями. Це дає змогу оцінити точність реалізації функцій та своєчасно виявити можливі помилки.

Належною практикою є проведення тестування незалежним користувачем, який не брав участі у розробці. Це дозволяє отримати об’єктивну оцінку, позбавлену упередженості.

У процесі тестування ПЗ застосовуються різні підходи, зокрема статичне, динамічне та дослідницьке тестування.

Статичне тестування

Статичне тестування передбачає аналіз коду та його артефактів без запуску програми. Основна мета — виявити потенційні проблеми ще на початкових стадіях розробки.

До методів статичного тестування належать:

рецензії та інспекції коду;

аналіз архітектурних рішень;

використання лінтерів;

перевірка відповідності стандартам оформлення коду;

пошук слабких місць у структурі безпеки програми.

Перевага статичного аналізу полягає в тому, що він дозволяє виявити значну кількість помилок до того, як вони почнуть впливати на роботу системи. Проте цей метод не може забезпечити повне покриття, адже не враховує поведінку програми під час її виконання. Тому статичне тестування зазвичай поєднується з динамічним.

Динамічне тестування

Динамічне тестування оцінює роботу програми під час її фактичного виконання. Система запускається, обробляє дані, взаємодіє з іншими процесами, і на основі цього аналізуються можливі помилки та невідповідності.

Основні напрями динамічного тестування:

функціональне тестування — перевірка відповідності функціоналу заявленим вимогам;

тестування продуктивності — оцінка швидкості роботи та часу реакції;

тестування навантаження — оцінка поведінки під високими навантаженнями;

тестування стійкості — визначення роботи в умовах помилкових або екстремальних сценаріїв.

Динамічне тестування може бути як автоматизованим, так і ручним. Його недоліком є те, що воно виявляє проблеми лише під час виконання програми. Найкращий результат забезпечує поєднання динамічного та статичного методів.

Дослідницьке тестування

Дослідницьке тестування базується на інтуїції, досвіді та творчому підході тестувальника. У цьому випадку не використовуються попередньо визначені сценарії — тестувальник взаємодіє з програмою в реальному часі та шукає нестандартні або непередбачувані ситуації.

Такий метод дозволяє знайти помилки, які складно виявити через формальні процедури. Разом із тим він менш структурований та менш відтворюваний, тому застосовується як доповнення до інших підходів.

Пасивне тестування

Пасивне тестування не передбачає активної взаємодії з програмою. Його мета — аналіз журналів, логів та інших артефактів роботи системи для виявлення аномалій або нестандартної поведінки.

Тестування «чорного» та «білого» ящика

Методи визначаються рівнем доступу до внутрішнього коду:

Метод чорного ящика

перевіряє відповідність функціоналу вимогам;

оцінює швидкодію під навантаженням;

тестує роботу з некоректними даними;

аналізує взаємодію з платформами, ОС, браузерами.

Метод білого ящика

перевіряє всі логічні гілки коду;

контролює взаємодію між модулями;

оцінює правильність реалізації окремих функцій;

визначає відсоток покриття коду тестами.

Організація тестування ПЗ «AiVirus»

Тестування проводилося як під час розробки, так і після завершення реалізації всіх модулів. Додатково система була протестована незалежним користувачем.

Для проведення тестування застосовувалися такі кроки:

перевірка працездатності механізму оновлення;

пошук загроз у вибраних файлах;

аналіз повного сканування системи;

тестування роботи режиму карантину.

Результати тестувань були задокументовані у вигляді скриншотів і описів виконаних операцій.

Першим етапом стала перевірка роботи кнопки «Оновлення» (рис. 3.17). Було виконано під’єднання до ресурсу з актуальними списками ключів. Оновлення розпочалося о 04:04 і завершилося о 04:33 — загалом 29 хвилин. За цей час система завантажила 487 нових або оновлених файлів (рис. 3.18).

Усі отримані дані були збережені у файлі «links_downloaded.txt» (рис. 3.19).

Функціонал оновлення відпрацював стабільно та повністю відповідав очікуваним результатам.

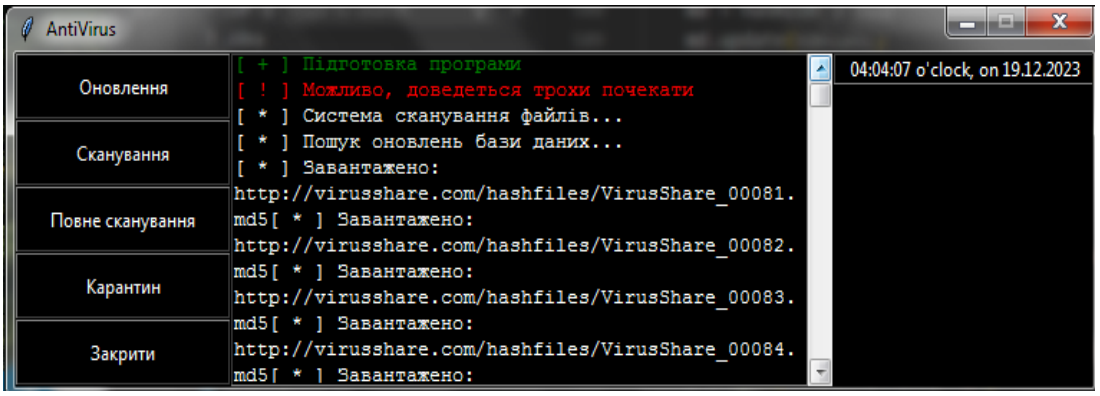

Рисунок 3.17 – Початок оновлення ключів

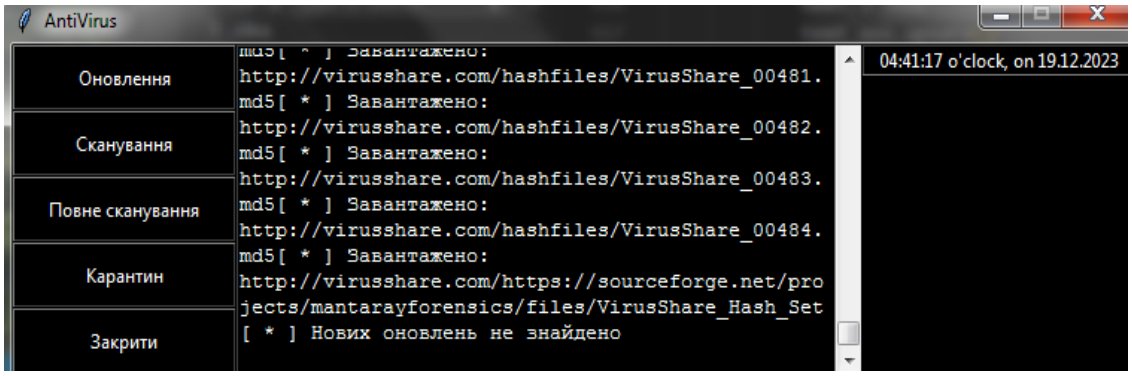

Рисунок 3.18 – Завершення тестування оновлень

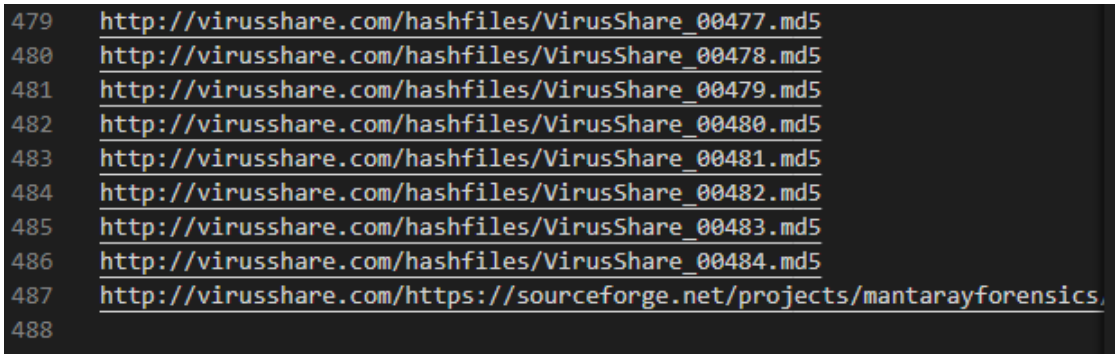

Рисунок 3.19 – Кількість оновлених ключів

Наступним етапом було тестування антивірусного сканера. Для перевірки попередньо було завантажено один із різновидів вірусу під назвою «Wizard of Windows» — шкідливу програму типу бекдор. Під час сканування одного інфікованого файлу система миттєво розпізнала загрозу, витративши на це лише 4,33 секунди. На рисунку 3.20 показано результат сканування, час виконання та повідомлення про виявлення зараження. Унаслідок перевірки інфікований файл було повністю видалено..

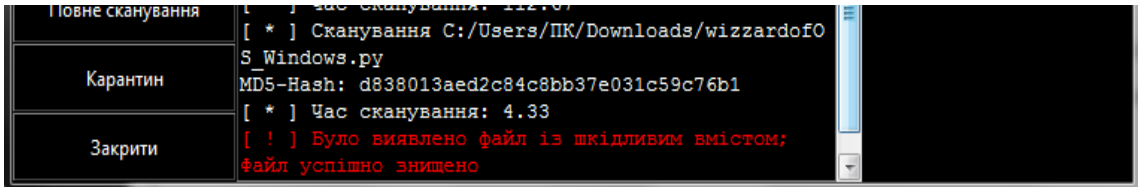

Рисунок 3.20 – Тестування сканування вибіркового файлу

Наступним етапом є тестування повного сканування всієї операційної системи. Недоліком цього режиму є тривалість процесу: залежно від кількості файлів і завантаженості системи, сканування може тривати від 40 хвилин до кількох годин. Після успішного завершення перевірки на екрані користувача з’явиться повідомлення про те, що операційна система повністю просканована та підготовлена до безпечної роботи. Цей результат показано на рисунку 3.21..

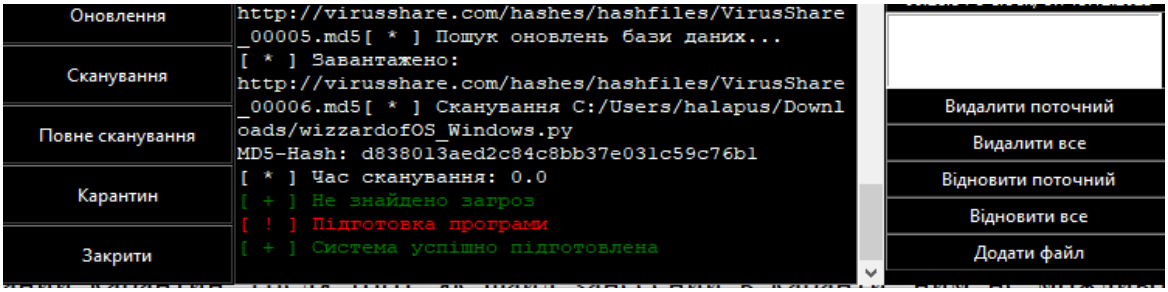

Рисунок 3.21 – Результат етапів сканування ОС

Повне сканування операційної системи виконується в тихому режимі. Це означає, що антивірус індексує та перевіряє файли у фоновому режимі, не відображаючи на екрані кожен відсканований документ. Усі оброблені файли записуються до бази даних у файл sf.txt. Тривалість сканування в тестовому випадку склала 54 хвилини.

Останнім етапом тестування стала перевірка функціональності карантину. Для цього було протестовано всі основні операції: додавання файлу до карантину, його видалення та відновлення. Усі функції спрацювали коректно, усі етапи були успішно пройдені та перевірені. Результат роботи карантину зображено на рисунку 3.22..

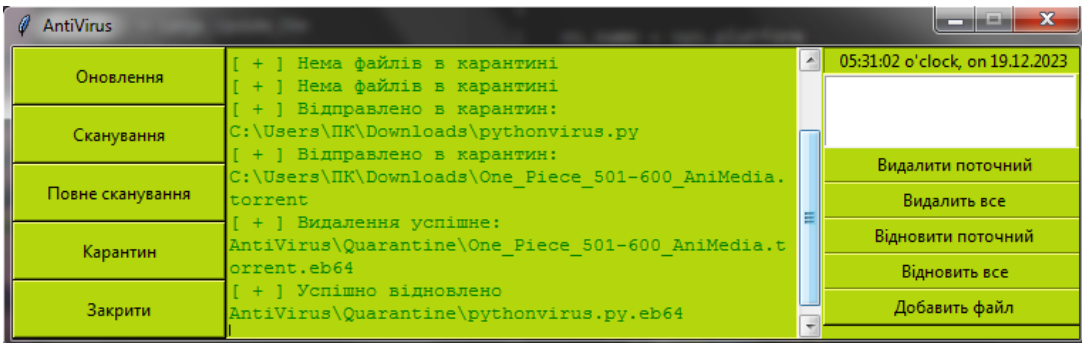

Рисунок 3.22 – Тестування функції карантину

При оцінюванні результатів тестування враховувалися такі ключові критерії: час виконання сканування, ефективність виявлення загроз, рівень захисту, а також зручність і якість користувацького інтерфейсу. Загальний

результат тестування виявився позитивним [4].

Програмний продукт «AiVirus» повністю відповідає всім заявленим вимогам. Під час перевірки на наявність шкідливого програмного забезпечення система миттєво виявляла та безповоротно видаляла загрози з операційної системи.

Роботу над подальшим удосконаленням продукту, зокрема розширенням функціональних можливостей та покращенням алгоритмів виявлення складніших і нових типів вірусів, можна й варто продовжувати. Проте на поточному етапі розроблений антивірусний засіб повністю готовий до практичної експлуатації в реальних умовах..

3.6 Рекомендації щодо впровадження та використання програмного засобу «AiVirus»

Для забезпечення стабільної та ефективної роботи антивірусного засобу рекомендується дотримуватися таких правил і рекомендацій:

1. Регулярне оновлення баз сигнатур. Найкраще виконувати оновлення щодня, оскільки списки сигнатур вірусів оновлюються розробниками кілька разів на добу. Часті оновлення значно підвищують рівень захисту.

2. Мінімальні системні вимоги до комп’ютера:

- Операційна система: Windows 7, 8.1, 10 або 11 (рекомендується 64-бітна версія).

- Оперативна пам’ять: не менше 8 ГБ.

- Постійний доступ до Інтернету (необхідний для автоматичного або ручного оновлення сигнатур).

3. Встановлення необхідного програмного забезпечення:

- Інтегроване середовище розробки Visual Studio Code.

- Мова програмування Python версії 3.9 (для Windows 7 підходять

Python 3.8 та старші, для Windows 10/11 — усі версії 3.9 і новіші).

4. Перевірка завантажених файлів. Після завантаження будь-якого файлу чи програми з невідомого або неперевіреного джерела обов’язково виконуйте ручне сканування перед відкриттям.

5. Повне сканування системи. Під час повного сканування антивірус працює в тихому (фоновому) режимі. Після завершення на екрані з’явиться повідомлення: «Система успішно підготовлена».

6. Використання карантину. Якщо файл викликає підозру, перемістіть його до карантину. Файли можуть зберігатися там необмежений час без можливості виконання.

7. Зникнення файлів. Якщо після сканування певний файл або програма зникли, це означає, що антивірус виявив у них шкідливий код і автоматично видалив. У такому разі рекомендується перевірити надійність джерела, з якого було завантажено файл.

8. Проблеми з запуском через термінал. Якщо програму не вдається запустити, перевірте відповідність версій Python, Visual Studio Code та операційної системи. Для перевірки встановленої версії Python у терміналі VS Code введіть команду:

python --version

і натисніть Enter.

9. Перегляд завантажених сигнатур. Для ознайомлення з переліком сигнатур, які були завантажені під час останнього оновлення, відкрийте файл `links_downloaded.txt` у кореневій папці програми.

Дотримання цих рекомендацій гарантує коректну роботу антивірусного засобу, мінімізує ймовірність помилок при першому запуску та забезпечує максимальний рівень захисту вашої системи.

ВИСНОВОК

У даній кваліфікаційній роботі було розроблено антивірусний засіб на основі сигнатурного методу виявлення шкідливого програмного забезпечення. Основною метою створеного програмного продукту є своєчасне виявлення та нейтралізація загроз до моменту інфікування операційної системи, а також можливість вибіркового сканування окремих файлів чи програм. Розробка виконана мовою програмування Python з використанням інтегрованого середовища Visual Studio Code.

У процесі виконання роботи було вирішено такі завдання:

- вивчено предметну область, класифікацію комп'ютерних загроз та принципи їх функціонування;

- проаналізовано методи та засоби захисту комп'ютерних інформаційних систем, а також механізми роботи комп'ютерних вірусів;

- проведено порівняльний аналіз сучасних антивірусних програм (Zillya! Total Security, Avast Free AiVirus, Avira Internet Security, Bitdefender та Windows Defender) за критеріями ефективності виявлення загроз, рівня захисту та зручності інтерфейсу;

- сформовано та реалізовано технічні вимоги до системи;

- розроблено програмний код антивірусного засобу;

- вдосконалено користувацький інтерфейс;

- проведено комплексне тестування програмного забезпечення;

- описано основні режими роботи засобу;

- підготовлено інструкцію користувача.

У результаті було створено повноцінний програмний засіб захисту, який реалізує один із ефективних методів запобігання несанкціонованому проникненню та інфікуванню комп'ютерних систем. Антивірус забезпечує як вибіркове сканування окремих файлів, так і повне сканування операційної системи, дозволяючи значно знизити ризик зараження, захистити

конфіденційні дані користувача та запобігти несанкціонованому доступу.

Розроблений програмний продукт пройшов успішне тестування та готовий до впровадження в реальних умовах, зокрема в товариствах з обмеженою відповідальністю.

Перспективи подальшого розвитку проєкту:

- реалізація автоматичного самостійного оновлення баз сигнатур;

- розширення функціональних можливостей та створення ще зручнішого й сучаснішого інтерфейсу;

- впровадження модуля «безпечні покупки в Інтернеті» та захисту під час роботи в мережі;

- інтеграція власного VPN-сервісу для шифрування трафіку.

Таким чином, створений антивірусний засіб є повноцінним, працездатним рішенням, яке може успішно використовуватися для захисту інформації та операційних систем від сучасних кіберзагроз, а також має значний потенціал для подальшого розвитку й комерційного застосування...

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Атаки типу Man-In-The-Middle: що треба знати кожному. Домени – перевірка та реєстрація доменів в Україні | Imena.ua. URL: https://www.imena.ua/blog/man-in-the-middle

2. Бібліотеки Python: потужні рішення у розробці ПЗ. FoxmindEd. URL: https://foxminded.ua/biblioteky-Python/.

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

4. 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.

5. Іванченко, І. М. Інформаційна система оцінювання знань в області тестування програмного забезпечення». Master's thesis, Сумський державний університет, 2021.https://essuir.sumdu.edu.ua/handle/123456789/81439.

6. Комп’ютерні моделі. Information Technologies and Learning Tools 44, № 6 (6 листопада 2014): 171–81. https://doi.org/10.33407/itlt.v70i2.2907.

7. Маліновська, О. О. «Система вибору методів захисту програмного забезпечення». Thesis, Чернігів, 2021. http://ir.stu.cn.ua/123456789/22647.

8. 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.

9. Швед, А. С. «Інформаційна технологія для автоматизації моніторингу програмного та апаратного забезпечення комп'ютерів локальної мережі». Master's thesis, Сумський державний університет, 2018. URL: http://essuir.sumdu.edu.ua/handle/123456789/72193.

10. Aleem S., Capretz L. F., Ahme F. Game development software engineering process life cycle: a systematic review. J. Softw. Eng. Res. Dev. 2016. Vol. 4.6. doi:10.1186/s40411-016-0032-7.

11. BSTU Laboratory of Artificial Neural Networks. URL: https://neuro.bstu.by/ai/To-dom/My_research/Paper-0-again/For-research/D-

mining/Anomaly-D/Intrusion-detection/taxonomy.pdf.

12. Dougan, Timothy, and Kevin Curran. «Man in the Browser Attacks.» International Journal of Ambient Computing and Intelligence 4, no. 1 (January 2012): 29–39. http://dx.doi.org/10.4018/jaci.2012010103.

13. Kovalenko, O. «Методи якісного аналізу та кількісної оцінки ризиків розробки програмного забезпечення». Системи управління, навігації та зв'язку. Збірник наукових праць 3, № 49 (3 липня 2018): 116–25. http://dx.doi.org/10.26906/sunz.2018.3.116.

14. Microsoft. Language support in visual studio code. Visual Studio Code - Code Editing. Redefined.

URL: https://code.visualstudio.com/docs/languages/overview

15. Stetsenko, Inna, та Viktoriia Savchuk. «Метод автоматизації тестування на проникнення вебатак». Technical sciences and technologies, № 1(19) (2020): 98–103. http://dx.doi.org/10.25140/2411-5363-2020-1(19)-98-103.

ДОДАТОК А

Код для програмного забезпечення

```
    global text_box

    match = False
    file = askopenfilename()
    start = time.time()
    text_box.insert(END, "[ * ] Сканування " + file + "\n")
    text_box.see(END)
    text_box.update()
    try:
        f = open(file, "rb")
        content = f.read()
        f.close()
        content = create_md5(content)
        text_box.insert(END, "MD5-Hash: " + content.decode("utf-8") + "\n")
        text_box.see(END)
        text_box.update()
    except MemoryError:
        text_box.insert(END, "[ - ] Неможливо створити MD5-Hash:\n----->MemoryError!\n", 'negative')
        text_box.insert(END, "[ ! ] Виберіть файли під 1 ГБ\n", "negative")
        text_box.tag_config('negative', foreground="red")
        text_box.see(END)
        text_box.update()
        return None
    except Exception as e:
        text_box.insert(END, "[ ! ] Не вдається впоратися з проблемою \ n [ ! ] Спробуйте ще раз / файл може бути пошкоджений\n", "negative")
        text_box.tag_config('negative', foreground="red")
        text_box.see(END)
        text_box.update()
        return None

    signatures = open(large_signatures, "rb")
    #runtime of a scan varies from system to system(time on the systems tested: 1s <= t <= 20s)
```

Рисунок А.1 – Код для сканування файлів на наявність вірусних сигнатур

```
    if content in signatures.read():#fastest solution
        signatures.close()
        match = True
    else:
        match = False
        signatures.close()
except MemoryError:
    try:
        signatures.close()
        signatures = open(large_signatures, "rb")
        if content in signatures.readlines():#again fast, but around 4 times slower than the fastest
            f.close()
            match = True
        else:
            signatures.close()
            match = False
    except MemoryError:
        signatures.close()
        signatures = open(large_signatures, "rb")
        while True:#slowest solution, but can read files sized over 2 GB
            tmp = signatures.readline()
            if tmp == b"":
                signatures.close()
                break
            if tmp == content:
                match = True
                signatures.close()
except:
```

Рисунок А.2 – Продовження коду для пошук файлів на наявність вірусних сигнатур

```
except:
    text_box.insert(END, "[-] Щось погане сталося під час виконання завдання \n", "negative")
    text_box.tag_config("negative", foreground="red")
    text_box.see(END)
    text_box.update()
    return None

text_box.insert(END, "[ * ] Час сканування: {0}\n".format(round(time.time()-start, 2)))
text_box.see(END)
text_box.update()
if match:
    quarantaene.encode_base64(file, file_to_quarantine)
    text_box.insert(END, "[ ! ] Загрози знайдені: {0}\n[ ! ] Файл був переміщений в карантин", "important")
    text_box.tag_config("important", foreground="red")
    text_box.see(END)
    text_box.update()
if not match:
    text_box.insert(END, "[ + ] Не знайдено загроз\n", "positive")
    text_box.tag_config("positive", foreground="green")
    text_box.see(END)
    text_box.update()
```

Рисунок А.3 – Код для виведення результатів на екран користувача

```
update_button = tkinter.Button(main, bg=bgc, fg=fgc, text = "Оновлення", command=lambda:button_action_handler("update_button"), height = hoe
update_button.grid(row = 0, column = 0)
scan_button = tkinter.Button(main, bg=bgc, fg=fgc, text = "Сканування", command=lambda:button_action_handler("scan_button"), height = hoehe,
scan_button.grid(row = 1, column = 0)
fullscan_button = tkinter.Button(main, bg=bgc, fg=fgc, text = "Повне сканування", command=lambda:button_action_handler("fullscan_button"), h
fullscan_button.grid(row = 2, column = 0)
quarantine_button = tkinter.Button(main, bg=bgc, fg=fgc, text = "Карантин", command=lambda:button_action_handler("quarantine_button"), heigh
quarantine_button.grid(row = 3, column = 0)
quit_button = tkinter.Button(main, bg=bgc, fg=fgc, text = "Закрити", command=lambda:button_action_handler("quit_button"), height = hoehe, wi
quit_button.grid(row = 4, column = 0, sticky="w")
b_delete = tkinter.Button(main, bg=bgc, fg=fgc, text = "Видалити поточний", height=0, width = 25, justify=CENTER)
b_delete_all = tkinter.Button(main, bg=bgc, fg=fgc, text = "Видалити все", height = 0, width = 25, justify=CENTER)
b_restore = tkinter.Button(main, bg=bgc, fg=fgc, text = "Відновити поточний", height=0, width = 25, justify=CENTER)
b_restore_all = tkinter.Button(main, bg=bgc, fg=fgc, text = "Відновити все", height = 0, width = 25, justify=CENTER)
b_add_file = tkinter.Button(main, bg=bgc, fg=fgc, text = "Додати файл", height = 0, width = 25, justify=CENTER)
b_delete.place(x = 570, y = 70)
```

Рисунок А.4 – Створення кнопки за допомогою бібліотеки tkinter