Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Пояснювальна записка

до магістерської кваліфікаційної роботи

на тему «Метод пошуку шкідливого програмного забезпечення»

08-20.МКР.011.00.000 ПЗ

Виконав: студент 2 курсу, групи 1БС-18м
Спеціальність 125 Кібербезпека
ОПП Безпека інформаційних і
комунікаційних систем

_____Новотарський О. Ю.

Керівник: к.т.н. ст. викл. каф. ЗІ

_____ Лукічов В.В.

Рецензент: к.т.н., доц., доц. кафедри ОТ

_____ Крупельницький Л. В.

Вінниця - 2019 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Освітньо-кваліфікаційний рівень магістр
Спеціальність 125 Кібербезпека
ОПП Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ, д. т. н., проф.
_____ В. А. Лужецький
_____ _____ 2019 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Новотарському Олексію Юрійовичу

1. Тема роботи: «Метод пошуку шкідливого програмного забезпечення» керівник роботи: Лукічов Віталій Володимирович, к.т.н. ст. викл. каф. ЗІ, затверджена наказом ректора ВНТУ№ 254 від 02.10.2019 р.
2. Строк подання студентом роботи _____ 2019 р.
3. Вихідні дані до роботи:
   - об'єкт захисту – файли на комп'ютері;
   - метод захисту – евристичний метод на основі сигнатур;
   - мова програмування C#;
   - операційна система сімейста Windows,
   - оперативна память – не менше 500 мб.
4. Зміст розрахунково-пояснювальної записки: Вступ. Аналіз літературних джерел. Розробка методу пошуку шкідливого програмного забезпечення шляхом прогнозування загроз. Розробка програмного застобу та експерементальне дослідження. Економічна частина. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу.

Схема методи виявлення шкідливого програмного забезпечення (плакат, А4). Загальна схема пошуку вірусів (плакат, А4). Схема функціонування програми (плакат, А4). Схема ієрархія класів нейронної мережі (плакат, А4). Схема алгоритму пошуку шкідливого програмного забезпечення (плакат, А4). Схема ресурсів програми (плакат, А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Лукічов В. В., к.т.н. ст. викл. каф. ЗІ		
2	Лукічов В. В., к.т.н. ст. викл. каф. ЗІ		
3	Лукічов В. В., к.т.н. ст. викл. каф. ЗІ		
4	Мацкевічус С. С., ст. викл. каф. ЕПВМ		

7. Дата видачі завдання _____ 2019 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Приміт ка
1	Аналіз завдання. Вступ	01.09.2019 – 04.09.2019	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	05.09.2019 – 5.09.2019	
3	Науково-технічне обґрунтування	15.09.2019 – 26.09.2019	
4	Розробка технічного завдання	27.09.2019 – 30.09.2019	
5	Розробка рішень	30.09.2019 – 12.10.2019	
6	Практична реалізація, моделювання, експериментування, результати	12.10.2019 – 10.11.2019	
7	Розробка розділу економічного обґрунтування доцільності розробки	11.11.2019 – 17.11.2019	
8	Аналіз виконання ТЗ, висновки	19.11.2019 – 24.11.2019	
9	Оформлення пояснювальної записки	25.11.2019 – 30.11.2019	
10	Попередній захист та доопрацювання МКР	26.11.2019 – 01.12.2019	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2019 – 10.12.2019	
12	Представлення МКР до захисту	11.12.2019 – 14.12.2019	
13	Захист МКР	16.12.2019 – 20.12.2019	

Студент _____ Новотарський О. Ю.
(підпис)

Керівник роботи _____ Лукічов В. В.
(підпис)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена розробці програмного засобу для забезпечення захисту комп'ютерів від шкідливого програмного забезпечення. Для пошуку шкідливого програмного забезпечення використано власний метод пошуку – евристичний метод на основі сигнатур шляхом прогнозування загроз, який дозволяє з певним рівнем співпадіння знаходити шкідливий програмний код у файлах за допомогою бази сигнатур.

Для успішної розробки програмного засобу проведено дослідження основних методів пошуку шкідливого програмного забезпечення, обґрунтовано вибір методу пошуку шкідливого коду у даній роботі, розроблено ряд схем і алгоритмів, здійснено програмну реалізацію. Засіб перевірено з метою доведення ефективності здійснюваного захисту.

ABSTRACT

The master's qualification is dedicated to the development of software to protect computers from malicious software. To find malware, you have used your own search method — a signature-based heuristic method of threat prediction that allows you to find malicious code in files using a signature database with some degree of coincidence.

In order to successfully develop the software, the basic methods of searching for malicious software were investigated, the choice of the method of finding the malicious code in this work was substantiated, a number of schemes and algorithms were developed, the software implementation was carried out. The tool is tested to prove the effectiveness of the protection.

ЗМІСТ

ВСТУП

Безпека програмного забезпечення є його властивістю функціонувати без прояву різноманітних негативних наслідків для конкретної комп'ютерної системи. Під рівнем безпеки програмного забезпечення (ПЗ) розуміється ймовірність того, що при заданих умовах у процесі його експлуатації буде отриманий функціонально придатний результат.

Нині відомі десятки тисяч комп'ютерних вірусів, які поширюються через мережу Інтернет по всьому світу. Необізнані користувачі персонального комп'ютера помилково завантажують програмні додатки з вірусами, що може призвести до непоправної шкоди операційній системі та іншим програмним додатках [1].

Кожен зловмисник, який намагається поцупити чи ушкодити інформацію, яка йому не належить або він не має права доступу до неї, повинен бути притянутий до кримінальної відповідальності за статтею 361 Кримінального кодексу України, яка говорить, що несанкціоноване втручання в роботу електронно-обчислювальних машин (комп'ютерів), автоматизованих систем, комп'ютерних мереж чи мереж електрозв'язку, що призвело до витоку, втрати, підробки, блокування інформації, спотворення процесу обробки інформації або до порушення встановленого порядку її маршрутизації, карається штрафом від шестисот до тисячі неоподатковуваних мінімумів доходів громадян або обмеженням волі на строк від двох до п'яти років, або позбавленням волі на строк до трьох років, з позбавленням права обіймати певні посади чи займатися певною діяльністю на строк до двох років або без такого та з конфіскацією програмних та технічних засобів, за допомогою яких було вчинено несанкціоноване втручання, які є власністю винної особи[2].

Тому було прийнято рішення, як спеціаліста з захисту інформаційних і комунікаційних систем покращити власний еврестичний метод пошуку вірусів за допомогою сигнатур та розробити антивірус для захисту користувачів комп’ютерних засобів.

Задачею антивірусних програм є знаходження комп'ютерних вірусів, а також небажаних (шкідливих) програм загалом та відновлення заражених (модифікованих) такими програмами файлів, а також для профілактики — запобігання зараження (модифікації) файлів чи операційної системи шкідливим кодом.

Об’єктом магістерської кваліфікаційної роботи є процес пошуку шкідливого програмного забезпечення.

Предметом магістерської кваліфікаційної роботи є методи пошуку шкідливого програмного забезпечення.

Метою магістерської кваліфікаційної роботи є покращення методів пошуку шкідливого программного забезпечення:

Для досягнення мети необхідно розв’язати такі задачі:

– проаналізувати методи пошуку вірусів за допомогою антивірусних програм;
– проаналізувати типи нейромереж.
– покращити власний еврестичний метод пошуку вірусів за допомогою сигнатур, який покращить процес пошуку вірусів в операційних системах;
– реалізувати програмний засіб з застосуванням евристичного алгоритму пошуку вірусів;
– протестувати розроблений програмний засіб.

1 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

З появою Інтернету мережеві і комунікаційні технології отримали сильний поштовх в своєму розвитку. Об'єми інформації через всілякі канали зв'язку сильно зросли за останні декілька років і ще будуть рости в майбутньому. Все це створює сприятливе середовище для бурхливого розвитку і шкідливих програм, тим більше на тлі поганої захищеності більшої частини мереж.

1.1 Аналіз антивірусних програм

Нині існує багато антивірусних засобів, найбільш популярними в нашій країні є такі комплекти: ЗАТ «Лабораторія Касперського», Avira, Norton Antivirus, Dr Web, AidsTest, Doctor Web, AVP.

До стандартної поставки антивірусноих комплектів АТ «ДіалогНаука» входять чотири програмних продукти: щотижня оновлюваний Aidstest, ревізор диска ADinf, лікуючий блок ADinf Cure Module і програма Doctor Web, що відслідковує і видаляє складно зашифровані та поліморфні віруси. У розширений варіант поставки комплекту входить апаратний комплекс Sheriff, що запобігає на апаратному рівні проникнення вірусів у систему.

Одним з найбільш популярних засобів проти вірусів є, як відомо, Aidstest, але, використовуючи його, потрібно пам'ятати, що він охороняє тільки від вірусів, з якими він вже знайомий. Для забезпечення більш ефективної безпеки використання Aidstest необхідно поєднувати з повсякденним використанням ревізора диска Adinf [].

Ревізор ADinf дозволяє виявити появу будь-якого вірусу, включаючи Stealth-віруси, віруси-мутанти і невідомі на сьогоднішній день віруси. При встановленій програмі ADinf Cure Module (лікуючий блок ревізора ADinf) можна видалити до 97% з них. ADinf бере під контроль всі дані вінчестера, куди можливе проникнення вірусу. Такий спосіб перевірок повністю виключає маскування Stealth-вірусів і забезпечує дуже високу швидкість перевірки диска.

Розширення ревізора ADinf - Cure Module (файл ADinfExt.exe) додатково підтримує базу даних, яка описує файли, що зберігаються на диску. У разі виявлення вірусу вона дозволяє негайно вилікувати машину.

Doctor Web бореться з відомими програмі поліморфними вірусами. Крім того, Doctor Web проводить евристичний аналіз файлів з метою виявлення невідомих вірусів, у тому числі складно зашифрованих і поліморфних вірусів.

Успіх такого аналізу - у середньому 82% [5]. Крім того програма може розпаковувати і перевіряти виконувані файли, оброблені архіваторами LZEXE, PKLite і Diet.

AVP

Антивірусний набір засобів, який є розширеною версією відомого антивірусного комплекту «Антивірус Касперський». У процесі роботи програми проводиться тестування на невідомі віруси. Також у комплект входить резидентна програма, що відстежує підозрілі дії, що здійснюються на комп'ютері, і що дає можливість проглядати карту пам'яті. Спеціальний набір утиліт допомагає виявляти нові віруси та розбиратися в них.

Norton Antivirus

Антивірусний засіб Norton Antivirus відноситься до засобів типу «встановив і забув». Всі необхідні параметри конфігурації і планові заходи (перевірка диска, перевірка нових і модифікованих програм, запуск Windows-утиліти Auto-Protect, перевірка boot-сектора перед перезавантаженням) встановлюються за замовчуванням. Програма сканування диска існує для операційних систем сімейства Windows. У числі інших Norton AntiVirus виявляє та знешкоджує навіть поліморфні віруси, а також успішно реагує на вірусоподібну активність і бореться з невідомими вірусами .

Проаналізувавши антивірусні програми потрібно проаналізувати статистичні дані виявлення вірусів, щоб дізнатись працездатнісь антивірусів, пересвідчитись в потребі їх використання та дослідити роботу найпоширеніших антивірусів.

За аналізом інформаційних джерел, було складено рейтинг з двадцяти загроз, які в 2017 році найчастіше виявлялись та знешкоджувались на комп'ютерах користувачів (рис. 1.1). В даний рейтинг також не входять програми класів Adware і Riskware[6] .

Перше місце займає DangerousObject.Multi.Generic (39,70%), що використовується для шкідливих програм, виявлених за допомогою хмарних технологій. Ці технології працюють, коли в антивірусних базах ще немає ні сигнатури, ні евристики для детектування шкідливої програми, але в хмарі антивірусної компанії вже є інформація про об'єкт.

	Название*	% уникальных атакованных пользователей**
1	DangerousObject.Multi.Generic	39,70
2	Trojan.Win32.Generic	27,30
3	Trojan.WinLNK.StartPage.gena	17,19
4	Trojan.Win32.AutoRun.gen	6,29
5	Virus.Win32.Sality.gen	5,53
6	Worm.VBS.Dinihou.r	5,40
7	Trojan.Script.Generic	5,01
8	DangerousPattern.Multi.Generic	4,93
9	Trojan-Downloader.Win32.Generic	4,36
10	Trojan.WinLNK.Agent.ew	3,42
11	Worm.Win32.Debris.a	3,24
12	Trojan.VBS.Agent.ue	2,79
13	Trojan.Win32.Autoit.cfo	2,61
14	Virus.Win32.Nimnul.a	2,37
15	Worm.Script.Generic	2,23
16	Trojan.Win32.Starter.lgb	2,04
17	Worm.Win32.Autoit.aly	1,97
18	Worm.Win32.Generic	1,94
19	HiddenObject.Multi.Generic	1,66
20	Trojan-Dropper.VBS.Agent.bp	1,55

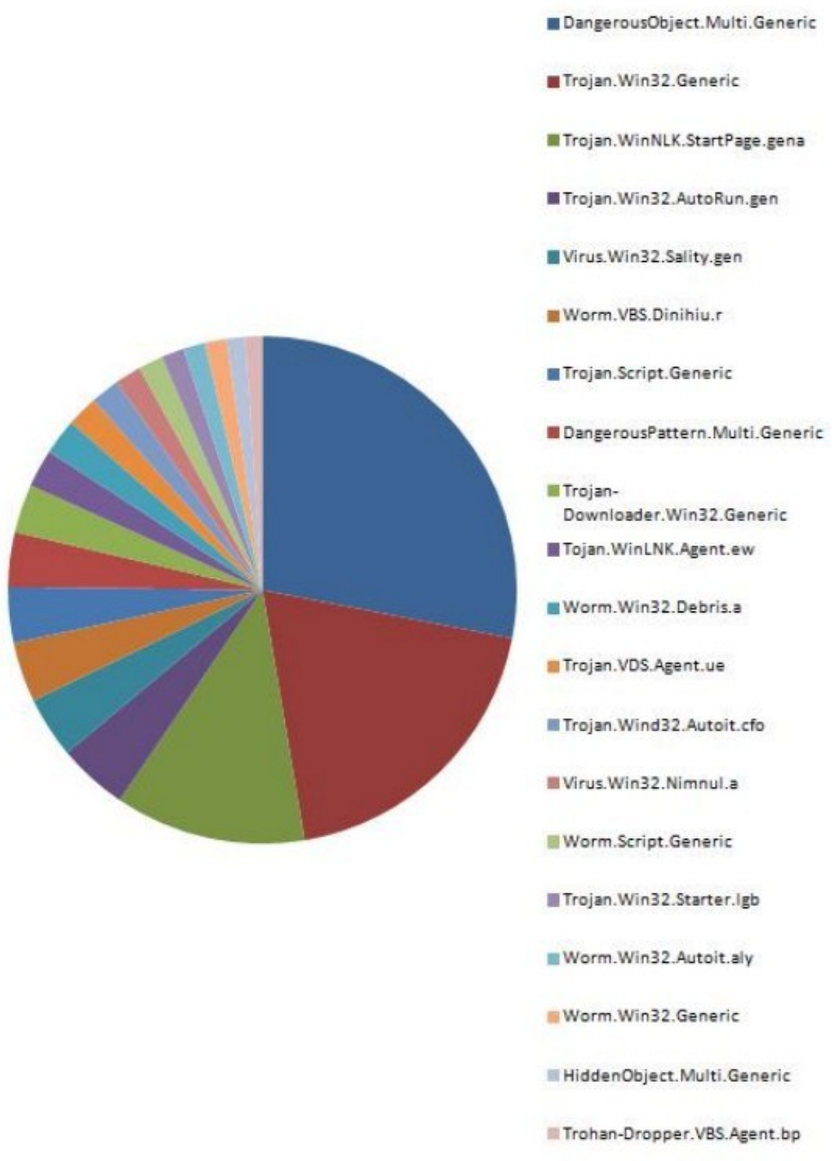


Рисунок 1.1 – Статистичні дані за 2019 рік

По суті, так фіксуються найновіші шкідливі програми. Продовжує падати частка вірусів: наприклад, Virus.Win32.Sality.gen в минулому році зустрічався у 6,69% користувачів, в 2018 - у 5,53%. Показник Virus.Win32.Nimnul.a в 2017 році - 2,8%, в 2013 - 2,37%. Присутній в рейтингу на двадцятому місці Trojan-Dropper.

VBS.Agent.bp є VBS-скрипт, який видає і зберігає на диск Virus.Win32.Nimnul[7].

Крім евристичних вірусів в ТОП 20 представлені хробаки, що поширюються на змінних носіях, і їх компонентів. Їх потрапляння в двадцятку обумовлено характером їх поширення і створенням безлічі копій. Хробак може продовжувати своє поширення протягом тривалого часу, навіть якщо його сервери управління вже не діють.

Країни, в яких комп'ютери користувачів піддавалися найбільшому ризику локального зараження.

Для кожної з країн було підраховано, наскільки часто протягом року користувачі в ній стикалися зі спрацьовуванням файлового антивіруса. Враховувалися детектовані об'єкти, знайдені безпосередньо на комп'ютерах користувачів або ж на знімних носіях, підключених до комп'ютерів, - флешках, картах пам'яті фотоапаратів, телефонів, зовнішніх жорстких дисках. Ця статистика відображає рівень зараженості персональних комп'ютерів в різних країнах світу [6].

ТОП 20 країн за рівнем зараженості комп'ютерів (рис. 1.2).

	Страна*	% уникальных пользователей**
1	Вьетнам	70,83
2	Бангладеш	69,55
3	Россия	68,81
4	Монголия	66,30
5	Армения	65,61
6	Сомали	65,22
7	Грузия	65,20
8	Непал	65,10
9	Йемен	64,65
10	Казахстан	63,71
11	Ирак	63,37
12	Иран	63,14
13	Лаос	62,75
14	Алжир	62,68
15	Камбоджа	61,66
16	Руанда	61,37
17	Пакистан	61,36
18	Сирия	61,00
19	Палестинская территория	60,95
20	Украина	60,78

Рисунок 1.2 – Розподіл країн за рівнем зараженості комп'ютерів

Після аналізу статистичних даних щодо пошуку вірусів за допомогою антивірусних програм побачимо, що користувачі є не захищеними від вірусів і

майже в кожного користувача була уражена операційна система вірусами. Користувач, який використовує антивірусні програми, не може бути впевнений на 100%, що на його комп`ютері немає вірусів. Велика кількість користувачів не використовує антивірусні програми, вони вважають, що ця проблема їх не стосується і насамперед, вони стають носіями вірусів і цим самим допомагають зловмисникам.

Потрібно розглянути методи пошуку вірусів за допомогою антивірусних програм та запропонувати метод покращення антивірусних програм.

1.2 Класифікація існуючих алгоритмів пошуку вірусів

В інформаційних джерелах питанню боротьби з вірусами приділяють неабияке значення.

На рисунку 1.3 зображена класифікація методів пошуку вірусів

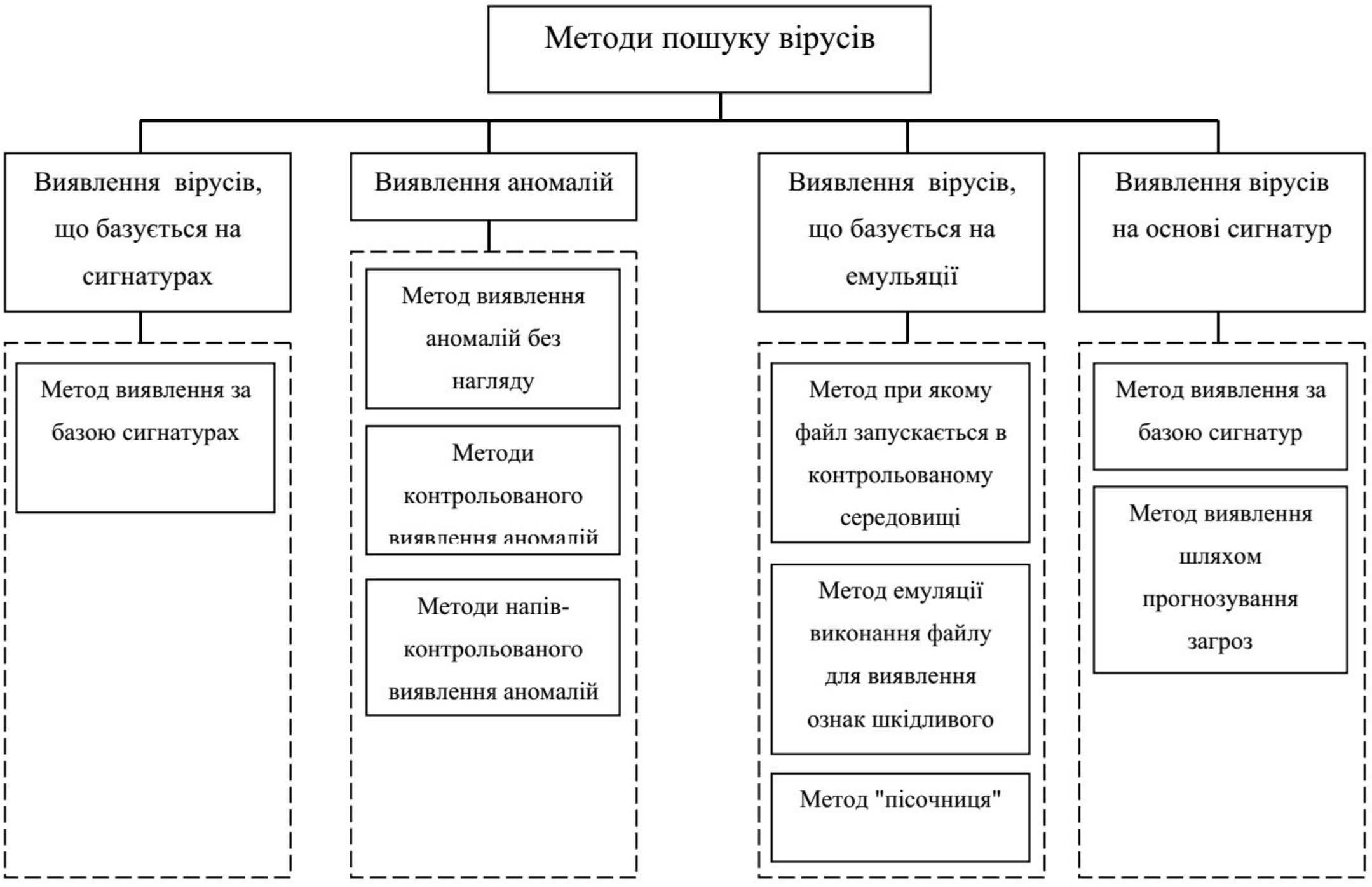


Рисунок 1.3 – Методи виявлення шкідливого програмного забезпечення

Зокрема, виділено такі основні способи запобігання вірусам:

– виявлення вірусів, що базується на сигнатурах;
– виявлення аномалій;
– виявлення вірусів, що базується на емуляції;
– виявлення вірусів евристичним методом на основі сигнатур.
– виявлення вірусів шляхом прогнозування загроз.

Оскільки темою магістерської кваліфікаційної роботи є реалізація певного способу боротьби з шкідливим програмним забезпечення, розглянемо детальніше кожен з цих методів.

1.3 Виявлення вірусів, засноване на сигнатурах

1.3.1 Можливості даного методу

Виявлення, що базується на сигнатурах - метод роботи антивірусів і систем виявлення вторгнень, при якому програмний засіб, переглядаючи файл або пакет, звертається до словника з відомими вірусами, складеним авторами програми. У разі відповідності будь-якої ділянки програмного коду, що переглядається, відомому коду (сигнатурі) вірусу в словнику, програма антивірус може зайнятися виконанням одного з наступних дій:

– видалити інфікований файл;
– відправити файл в «карантин» (тобто, заборонити йому винонуватись, з метою недопущення подальшого поширення вірусу).
– спробувати відновити файл, видаливши сам вірус з тіла файлу.

Для досягнення тривалого успіху при використанні цього методу необхідно періодично поповнювати словник відомих вірусів новими сигнатурами (в основному в онлайновому режимі). Технічно досвідчені користувачі, виявивши «живцем» новий вірус, можуть вислати заражений файл розробникам антивірусних програм, які включать потім новий вірус у словник.

Антивірусні програми, створені на основі методу відповідності визначенню вірусів у словнику, переглядають файли тоді, коли комп'ютерна

система створює, відкриває, закриває або посилає файли по електронній пошті. Таким чином, віруси виявляються відразу ж після занесення їх в комп'ютер і до того, як вони зможуть завдати шкоди. Треба відзначити, що системний адміністратор може скласти певний графік для антивірусної програми, згідно з яким можуть переглядатися всі файли на жорсткому диску [8].

Хоча антивірусні додатки, створені на основі пошуку відповідності визначенню вірусу у словнику, за звичайних обставин можуть досить ефективно перешкоджати спалахам зараження комп'ютерів, автори шкідливого програмного забезпечення намагаються триматися на півкроку попереду таких антивірусних засобів, створюючи «олігоморфичні», «поліморфічні» і найновіші «метаморфічні» віруси, в яких деякі частини ділянки коду повністю перезаписуються, модифікуються, шифруються або спотворюються так, щоб виявлення збігу було неможливим з визначенням в словнику вірусів.

1.3.2 Створення і розподіл сигнатур

Сигнатури антивірусних засобів створюються в результаті кропіткого аналізу декількох копій файлу, що належить одному вірусу. Сигнатура повинна містити унікальні рядки з цього файлу, настільки характерні, щоб можна було гарантувати мінімальну можливість помилкового спрацьовування - головний пріоритет будь-якої антивірусної компанії.

Розробка сигнатур – ручний процес, важко піддається автоматизації. Незважаючи на велику кількість досліджень, присвячених автоматичній генерації сигнатур, наростаючий поліморфізм (і «метаморфізм») вірусів і атак, роблять синтаксичні сигнатури безглуздими. Антивірусні компанії вимушені випускати велику кількість сигнатур для всіх варіантів одного і того ж вірусу, і якби не закон Мура [], жоден сучасний комп'ютер вже не зміг би завершити сканування великої кількості файлів з такою кількістю сигнатур в розумний час. Так, в березні 2006 року сканера Norton Antivirus було відомо близько 72 131 вірусів, а база програми містила близько 400 000 сигнатур [8].

У нинішньому вигляді бази сигнатур змушені поповнюватися регулярно, оскільки більшість антивірусів не в змозі виявляти нові віруси самостійно. Будь-який власник ПЗ, що базується на сигнатурах, приречений на регулярну залежність від оновлення сигнатур, що становить основу бізнес-моделі виробників антивірусів і систем обробки вірусів.

Своєчасна доставка оновлених сигнатур до користувачів також є серйозною проблемою для виробників ПЗ. Сучасні віруси поширюються з такою швидкістю, що до моменту випуску сигнатури та доставки її на комп'ютер користувачів епідемія вже може досягти своєї вищої точки і охопити весь світ. За опублікованими даними доставка сигнатури займає від 11 до 97 годин в залежності від виробника, в той час, як теоретично вірус може захопити весь інтернет менше, ніж за пів хвилини.

У більшості ПЗ такого типу бази сигнатур є ядром продукту – найбільш трудомісткою і найцінішою частиною. Саме тому більшість поставщиків вважає за краще тримати свої сигнатури закритими, хоча і в цій області існує певна кількість відкритого ПЗ (напр., ClamAV), а також дослідження по зворотній розробці закритих сигнатур. Virus Bulletin регулярно публікував сигнатури нових вірусів аж до 2001 року [8].

1.3.3 Основні характеристики сигнатурного методу

Програми, що реалізують цей метод, мають такі властивості:

- дозволяють визначати конкретну атаку з високою точністю і малою часткою помилкових викликів;
- беззахисні перед поліморфними вірусами і зміненими версіями того ж вірусу;
- вимагають регулярного і вкрай оперативного оновлення;
- вимагають кропіткого ручного аналізу вірусів;
- не можуть виявити будь-які нові атаки.

Метод евристичного сканування покликаний покращити здатність сканерів застосовувати сигнатури і виявляти модифіковані віруси в тих

випадках, коли сигнатура збігається з тілом невідомої програми не на 100%. Дана технологія, застосовується в сучасних програмах дуже обережно, так як може підвищити кількість помилкових спрацьовувань.

1.4 Виявлення вірусів, основане на емуляції

Виявлення, засноване на емуляції – метод роботи антивірусного програмного засобу, при якому підозрілий файл запускається в ретельно контрольованому середовищі, або емулюється його виконання з метою виявлення ознак шкідливого коду, які з'являються тільки під час виконання програми

Деякі антивіруси намагаються імітувати початок виконання коду кожної нової запущеної на виконання програми перед тим як передати їй управління. Якщо програма використовує само змінюваний код або проявляє себе як вірус, наприклад негайно починає шукати інші exe-файли, така програма буде вважатися шкідливою, здатною заразити інші файли. Однак цей метод теж має великою кількістю помилкових попереджень.

Переваги даного методу такі.

У деяких випадках, емуляція дозволяє ефективно протистояти таким технологіям, як поліморфізм шкідливих програм, що досягається за рахунок оцінювання здійснюваних дій, але не програмного коду. В даний час існує велика кількість платних і безкоштовних сервісів для аналізу невідомого програмного забезпечення. Ці сервіси використовують методи емуляції для протоколювання подій, що відбуваються в тестовій системі [10].

Недоліки емуляції такі.

Недоліком емуляції є високе споживання системних ресурсів, що негативно впливає на продуктивності комп'ютера. Тому на сьогоднішній день емуляція не є основною технологією антивірусного програмного забезпечення, помітно поступаючись сучасним проактивним методам антивірусного захисту (наприклад, HIPS).

Ще один метод виявлення вірусів включає в себе використання «пісочниці» (найчастіше заснованої на віртуальній машині). Пісочниця імітує операційну систему і запускає виконуваний файл в цій імітованій системі. Після виконання програми антивірус аналізує вміст пісочниці на присутність будь-яких змін, які можна кваліфікувати, як вірус. Через те, що швидкодія системи знижується і потрібен досить тривалий проміжок часу для виконання програми, антивірусні програми, побудовані за цим методом, зазвичай використовуються тільки для сканування за запитом користувача.

Слід зазначити, що ефективність даних програмних засобів набагато вища, ніж у всіх інших, але і витрати на їх роботу також високі. Без сумніву, ці програми представляють інтерес для фахівців, які займаються питаннями комп'ютерної безпеки та відновленням даних після несанкціонованого доступу в комп'ютер користувача або атак на сервер [11].

1.5 Класифікація архітектур нейромереж

Штучна нейронна мережа – це система з'єднаних і взаємодіючих між собою простих процесорів (штучних нейронів). Такі процесори зазвичай досить прості. Кожен процесор аналогічної мережі має справу тільки з сигналами, які він періодично отримує, і сигналами, які він періодично відправляє іншим процесорам. І, тим не менше, будучи з'єднаними в досить велику мережу з керованою взаємодією, такі окремо прості процесори разом здатні виконувати досить ресурсозатратні завдання. Є різні методи їх використання, а саме:

– З точки зору машинного навчання, нейронна мережа являє собою окремий випадок методів розпізнавання образів, дискримінантного аналізу, методів кластеризації.

– З математичної точки зору, навчання нейронних мереж - це багато параметричне завдання нелінійної оптимізації.

– З точки зору кібернетики, нейронна мережа використовується в задачах адаптивного управління і як алгоритми для робототехніки.

– З точки зору розвитку обчислювальної техніки та програмування, нейронна мережа - спосіб вирішення проблеми ефективного паралелізму.

– З точки зору штучного інтелекту є основою філософської течії коннекціонізма і основним напрямком в структурному підході з вивчення можливості побудови природного інтелекту за допомогою комп'ютерних алгоритмів.

Нейронні мережі не створюються в звичному сенсі цього слова, вони навчаються. Можливість навчання - одна з головних переваг нейронних мереж перед традиційними алгоритмами. Технічне навчання полягає в знаходженні коефіцієнтів зв'язків між нейронами. В процесі навчання нейронна мережа здатна знаходити складні залежності між вхідними даними і вихідними, а також виконувати узагальнення. Це означає, що в разі успішного навчання мережа може повертати достовірний результат на підставі даних, які були відсутні в навчальній вибірці, а також неповних та / або «зашумлених», частково спотворених даних.

Типи нейромереж:

– Перцептрон
– Згорткова нейронна мережі
– Адаптивна резонансна теорія
– Нейронна мережа Хопфілда
– Самоорганізована карта Кохонена

1.5.1 Перцептрон

Перцептрон являється мережою, що складається з S-, A- та R-елементів,

зі змінною матрицею взаємодії W що визначається послідовністю минулих станів активності мережі.

Важливою характеристикою будь-якої нейронної мережі є здатність до навчання. Процес навчання є процедурою налаштування ваг та порогів з метою зменшення різниці між бажаними та отримуваними векторами на виході.

Розенблат намагався класифікувати різні алгоритми навчання нейромережі перцептрон, називаючи їх системами підкріплення.

Система підкріплення — це певний набір правил, на підставі яких можна змінювати з плином часу матрицю взаємодії перцептрону, які складається з двох частин:

– Якщо два нейрони з обох боків синапсу (з'єднання) активізуються одночасно (тобто синхронно), то міцність цього з'єднання зростає.

– Якщо два нейрони з обох боків синапсу активізуються асинхронно, то такий синапс послаблюється або взагалі відмирає.

Класичний метод навчання перцептрону – це метод корекції помилки. Він являє собою такий вид навчання з учителем, при якому вага зв'язку є незмінним до тих пір, поки поточна реакція перцептрона залишається правильною. При появі неправильної реакції вага змінює своє знчення на одиницю, а знак (+/-) визначається протилежним від знаку помилки[12].

Припустимо, потрібно навчити перцептрон розділяти два класи об'єктів так, щоб при відображенні об'єктів першого класу вихід перцептрона був позитивний (+1), а при відображенні об'єктів другого класу — негативним (-1). Для цього виконаємо наступний алгоритм:

1. Випадково вибираємо пороги для А-елементів та встановлюємо зв'язки S-A (далі вони не змінюватимуться).

2. Початкові коефіцієнти вважаємо рівними нулеві.

3. Пред'являємо навчальну вибірку: об'єкти із зазначенням класу, до якого вони належать.

4. Показуємо перцептронові об'єкт першого класу. При цьому деякі А-елементи збудяться. Коефіцієнти, що відповідають цим збудженням елементів, збільшуємо на 1.

5. Пред'являємо об'єкт другого класу, і коефіцієнти тих А-елементів, які збудилися при цьому показі, зменшуємо на 1.

6. Обидві частини кроку 3 виконаємо для всієї навчальної вибірки. В результаті навчання сформуються значення вагів зв'язків .

Теорема збіжності перцептрону, показує, що елементарний перцептрон, навчений за таким алгоритмом, незалежно від початкових станів вагових коефіцієнтів і послідовності появи стимулів завжди призведе до досягнення рішення за скінченний проміжок часу.

Крім класичного методу навчання нейромережі перцептрон, Розенблат також ввів поняття про навчання без учителя, запропонувавши наступний спосіб навчання

Альфа-система підкріплення – це система підкріплення, за якої всі ваги активних зв'язків, що ведуть до елемента, змінюються на однакову величину r, а ваги неактивних зв'язків за цей час не змінюються.

Перцептрон може бути використано, для апроксимації функцій, для задачі прогнозування (й еквівалентної їй задачі розпізнавання образів), що вимагає високої точності, та задачі керування агентами, що вимагає високої швидкості навчання.

У практичних задачах від перцептрона вимагатиметься вибір більш ніж з двох варіантів, а отже, на виході в нього має бути більш ніж один R-елемента. Характеристики таких систем не відрізняються суттєво від характеристик елементарної нейромережі перцептрона.

1.5.2 Згорткова нейрона мережа

Згорткова нейрона мережа – спеціальна архітектура штучних нейронних мереж, яка націлена на ефективне розпізнавання образів, входить до складу технологій глибокого навчання. Використовує деякі особливості зорової кори, в якій були відкриті прості клітини, що реагують на прямі лінії під різними кутами, і складні клітини, реакція яких пов'язана з активацією набору простих клітин. Таким чином, ідея згорткових нейронних мереж полягає в чергуванні згорткових шарів і субдискритизуючих шарів. Структура мережі - односпрямована, принципово багатошарова. Для навчання використовуються

стандартні методи, найчастіше використовується метод зворотного поширення помилки. Функція активації нейронів – будь-яка, за вибором дослідника.

У звичайному перцептроні, який являє собою повнозв'язну нейронну мережу, кожен нейрон пов'язаний з усіма нейронами попереднього шару, причому кожний зв'язок має свій персональний ваговий коефіцієнт. У згорткової нейронної мережі в операції згортки використовується обмежена матриця ваг невеликого розміру, яку «рухають» по всьому оброблюваного шару формуючи після кожного зсуву сигнал активації для нейрона наступного шару з аналогічною позицією. Тобто для різних нейронів вихідного шару використовуються одна й та ж матриця ваг, яку також називають ядром згортки. Її інтерпретують як графічне кодування певної ознаки, наприклад, наявність похилої лінії під певним кутом. Тоді наступний шар, що виходить в результаті операції згортки такою матрицею ваг, показує наявність даної ознаки в оброблюваному шарі та її координати, формуючи так звану карту ознак. Природно, в згортковій нейронній мережі набір ваг не один, а ціла гама, що кодує елементи зображення. При цьому такі ядра згортки не закладаються дослідником заздалегідь, а формуються самостійно шляхом навчання нейромережі класичним методом зворотного поширення помилки. Прохід кожним набором ваг створює власний примірник карти ознак, роблячи нейронну мережу багатоканальною. При переборі шару матрицею ваг її пересувають зазвичай не на повний крок, а на невелику відстань. Так, наприклад, при розмірності матриці ваг 5 × 5 її зрушують на один або два нейрона замість п'яти, щоб не пропустити шукану ознаку[13].

Операція субдискритизації, виконує зменшення розмірності сформованих карт ознак. У даній архітектурі нейромережі вважається, що інформація про факт наявності шуканої ознаки важливіше точного знання його координат, тому з кількох сусідніх нейронів карти ознак вибираються максимальні і приймаються за один нейрон ущільненої карти ознак меншої розмірності. За рахунок цієї операції, крім прискорення подальших обчислень, мережа стає більш варіативною до масштабу вхідного зображення.

Мережа складається з великої кількості шарів. Після початкового шару сигнал проходить серію згорткових шарів, в яких чергується власне згортка і субдискритизація. Чергування шарів створювати складати карти ознак з карт ознак, на кожному наступному шарі карта зменшується в розмірі, але кількість каналів збільшується. На практиці це означає здатність розпізнавання складних ієрархій ознак. Зазвичай після проходження декількох шарів карта ознак перетворюється в вектор або навіть скаляр, але таких карт ознак стають сотні. На виході згорткових шарів мережі додатково встановлюють кілька шарів повнозв’язної нейронної мережі, на вхід якого подаються кінцеві карти ознак.

1.5.3 Нейронна мережа Хопфілда

Нейронна мережа Хопфілда – повнозв’язна нейронна мережа із симетричною матрицею зв'язків. В процесі роботи динаміка таких мереж сходиться до одного з положень рівноваги. Ці положення рівноваги визначаються заздалегідь в процесі навчання, вони є локальними мінімумами функціонала, званого енергією мережі. Така мережа може використовуватись як автоасоціативна пам'ять, як фільтр, а також для вирішення деяких завдань оптимізації. На відміну від багатьох нейромереж, що працюють до отримання відповіді через певну кількість тактів, мережі Хопфілда працюють до досягнення певної рівноваги, коли наступний стан мережі дорівнює попередньому: початковий стан є вхідним образом, а при рівновазі отримують вихідний образ[14].

Нейронна мережа Хопфілда влаштована так, що її відгук на запам’ятовані m еталонних образів складають самі ці образи, а якщо образ трохи спотворити і подати на вхід, він буде відновлений і в вигляді відгуку буде отримано оригінальний образ. Таким чином, мережа Хопфілда здійснює корекцію помилок і перешкод.

Мережа Хопфілда одношарова і складається з N штучних нейронів. Кожен нейрон системи може приймати на вході і на виході одне з двох станів: 1 або -1.

Через їх біполярну природу нейронні мережі Хопфілда іноді називають спінами.

Кожен нейрон пов'язаний з усіма іншими нейронами.

В процесі навчання формується вихідна матриця W, яка запам'ятовує m еталонних образів - N-мірних бінарних векторів ці образи під час експлуатації мережі будуть висловлювати відгук системи на вхідні сигнали, або інакше - остаточні значення виходів після серії ітерацій.

В мережі Хопфілда матриця зв'язків є симетричною, а діагональні елементи матриці є рівними нулю, що виключає ефект впливу нейрона на самого себе і є необхідним для нейромережі Хопфілда, але не достатньою умовою стійкості в процесі роботи мережі. Достатнім є асинхронний режим роботи мережі. Подібні властивості визначають тісний зв'язок з реальними фізичними речовинами, званими спінові стіклами.

Матриця взаємодій зберігається на самих нейронах у вигляді терезів при зв'язках нейронів з іншими нейронами.

Так наприклад, якщо вхідний сигнал визначається 10 параметрами, то нейронна мережа Хопфілда формується з одного рівня з 10 нейронами. Кожен нейрон зв'язується з усіма іншими 9-ю нейронами, таким чином в мережі утворюється 90 (10 х 9) зв'язків. Для кожного зв'язку визначається ваговий коефіцієнт. Всі ваги зв'язків і утворюють матрицю взаємодій, яка заповнюється в процесі навчання.

Алгоритм навчання нейромережі Хопфілда відрізняється від таких класичних алгоритмів навчання перцептронів, як метод зворотного поширення помилки або метод корекції помилки. Відмінність полягає в тому, що замість послідовного наближення до потрібного стану обчислення помилок, всі коефіцієнти матриці розраховуються за однією формулою, за один цикл, після чого нейромережа відразу готова до роботи.

У мережі Хопфілда є зворотні зв'язки і тому потрібно вирішувати проблему стійкості. Ваги між нейронами в нейромережі Хопфілда можуть розглядатися у вигляді матриці взаємодій. Нейромережа з зворотними зв'язками

є стійкою, якщо її матриця симетрична і має нулі на головній діагоналі. Є багато стійких систем іншого типу, наприклад, всі мережі прямого поширення, а також сучасні рекурентні мережі Джордана і Елмана, для яких не обов'язково виконувати умову на симетрію. Але це відбувається внаслідок того, що на зворотні зв'язки накладені інші обмеження. У разі нейромережі Хопфілда умова симетричності є необхідним, але не достатньою, тому що на досягнення стійкого стану впливає ще й режим роботи мережі. Тільки асинхронний режим роботи нейромережі гарантує досягнення стійкого стану мережі, в синхронному випадку можливо нескінченне перемикання між двома різними станами.

1.5.4 Самоорганізована карта Кохонена

Самоорганізована карта Кохонена – нейронна мережа з навчанням без учителя, що виконує завдання візуалізації і кластеризації. Ідея мережі запропонована фінським вченим Т. Кохоненом. Є методом проектування багатовимірних просторів в простір з більш низькою розмірністю (найчастіше, двовимірне), застосовується також для вирішення завдань моделювання, прогнозування, виявлення наборів незалежних ознак, пошуку закономірностей у великих базах даних, розробці комп'ютерних ігор, квантування квітів до їх обмеженому числу індексів в колірній палітрі: при друку на принтері і раніше на ПК або ж на приставках з дисплеєм зі зниженим числом квітів, для архиваторов або відео-кодеків[15].

Самоорганізована карта складається з певних компонентів, які називаються вузлами або нейронами. Їх кількість задається аналітиком. Кожен з вузлів описується двома векторами. Перший вектор ваги m, має таку ж розмірність, що і вхідні дані. Другий - вектор r, що являє собою координати вузла на карті. Карта Кохонена відображається візуально за допомогою осередків прямокутної або шестикутної форми, остання застосовується частіше, оскільки відстані між центрами суміжних осередків однакові, що підвищує коректність візуалізації карти.

Спочатку відома тільки розмірність вхідних даних, по ній певним чином будується початковий варіант карти. В процесі навчання вектори ваг вузлів наближаються до вхідних даних. Для кожного спостереження обирається найбільш схожий по вектору ваги вузол, і значення його вектора ваги наближається до спостереження. Також до спостереження наближаються вектори ваг декількох вузлів, розташованих поруч, таким чином якщо в вхідних даних два спостереження були схожі, на карті їм будуть відповідати близькі вузли. Циклічний процес навчання, перебираючий вхідні дані, закінчується після досягнення картою допустимої похибки, або після здійснення заданої кількості ітерацій. В результаті навчання карта Кохонена буде класифікувати вхідні дані на кластери і візуально відображати багатовимірні вхідні дані в двовимірній площині, розподіляючи вектори близьких ознак в сусідні осередки та розфарбовуючи їх в залежності від аналізованих параметрів нейронів.

В результаті роботи алгоритму виходять такі карти:

– карта входів нейронів - візуалізує внутрішню структуру вхідних даних шляхом підстроювання ваг нейронів карти. Зазвичай використовується декілька карт входів, кожна з яких відображає один з них і розфарбовується в залежності від ваги нейрона. На одній з карт певним кольором позначають область, в яку включаються приблизно однакові входи для проаналізованих прикладів.

– карта виходів нейронів - візуалізує модель взаємного розташування вхідних прикладів. Окреслені області на карті являють собою кластери, що складаються з нейронів зі схожими значеннями виходів.

– спеціальні карти - це карта кластерів, отриманих в результаті застосування алгоритму самоорганізується карти Кохонена, а також інші карти, які їх характеризують.

1.6 Обгрунтування вибору архітектури нейромережі

Проаналізувавши архітектури нейромереж стає зрозуміло, що вони поділяються за принципом використання, а саме:

– Розпізнавання образів і класифікація: перцептрон, згорткові нейронні мережі, мережі адаптивного резонансу;

– Прийняття рішень і управління: перцептрон;

– Кластеризація: перцептрон, самоорганізована карта Кохонена;

– Прогнозування: перцептрон;

– Апроксимація: перцептрон;

– Стиснення даних і асоціативна пам'ять: перцептрон, нейронна мережа Хопфілда;

– Аналіз даних: перцептрон, самоорганізована карта Кохонена.

Кожна архітектура є оптимальною для кожного роду використання.

Так як алгоритм повинен прогнозувати шкідливе програмне забезпечення в файлах, буде доцільно обрати нейромережу з архітектурою прогнозування.

Тому було прийняте рішення обрати нейромережу Перцептрон, так як вона належить до архітектури прогнозування та є універсальною мережою, яка може підійти для будь якої задачі.

Проаналізувавши архітектури мереж та обравши одну з них можна перейти до реалізації алгоритму програмного засобу .

2 РОЗРОБКА МЕТОДІВ ПОШУКУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Обравши модель нейромережі потрібно перейти до реалізації алгоритму прогнозування загроз за допомогою нейромережі та реалізації алгоритму пошуку вірусів еврестичним методом на основі сигнатур та реалізувати програмний засіб для пошуку шкідливого програмного забезпечення.

2.1 Розробка алгоритму нейромережі

Потрібно дізнатися чи файл є ураженим шкідливим програмним забезпеченням на основі одного параметра – кількості символів підряд шкідливого програмного забезпечення. Потрібно створити нейрон, який буде порушуватися в тих випадках, коли ймовірність шкідливого програмного забезпечення становить понад 50%.

Нейрон має один рецептор, пов'язаний з кількостю символів підряд шкідливого програмного забезпечення. Крім того, додаємо один bias член, який буде відповідати за зрушення. Наприклад, хоч ці числа й лінійно нероздільні, але приблизна межа, найкраще розділяє гіперповерхность, між ними розташовується в кількості символів підряд 18. Число було обрано випадковим чином для подальшого навчання нейромережі, яка скорегує кількість та відсоток до найбільш оптимального результату, тобто при якому відсоток похибки буде найменший.

Нейромережа повинна давати значення ймовірності покупки менше 0.8 при кількості символів підряд до 18, більше 0.5 для більшої кількості символів підряд. Функція активації, повертає значення більше 0.5 для позитивних аргументів і менше 0.5 для - негативних. Позитивний аргумент вважається, що файл уражений шкідливим програмним забезпеченням, а негативний, що файл вважається не ураженим шкідливим програмним забезпеченням. Значить, потрібна можливість зрушувати цю функцію активації на якесь порогове значення. При цьому очікується швидкість перелому функції активації, яка б

найкраще відповідала навчальній вибірці, так як від коефіцієнтів матриці ваг буде залежати ступінь збудження кожного нейрона як функція від вектора ознак x, і відповідно ймовірність приналежності елемента з таким вектором ознак до цього класу.

Щоб змістити функцію f (x) вправо, наприклад на 18, потрібно відняти 18 з її аргументу f (x-18). При цьому потрібно отримати слабкий перегин функції, помноживши аргумент на 0.25, отримавши таку функцію f (0.25 (x-24)). Розкриваючи дужки, отримаємо (вираз 2.1):

$$\sigma(z) = \frac{1}{1+e^{-0,25(x-18)}} = \frac{1}{1+ e^{-(0,25x-4.5)}} \equiv \frac{1}{1+ e^{-(wx+b)}} \quad (2.1)$$

Шуканий коефіцієнт матриці ваг w = 0.25, а зрушення b = -4.5. Вважаємо, що b це нульовий коефіцієнт матриці ваг (w0 = b) в тому випадку, якщо для будь-якого прикладу нульовим ознакою завжди буде одиниця (x0 = 1). Тоді, наприклад п'ятнадцятий «векторизований» файл з кількістю символів підряд в 18, представлений як x (15) = {x (15) 0, x (15) 1} = [1, 30], міг би бути ураженим шкідливим прогшрамним забезпеченням з ймовірністю 68%.

Проаналізувавши коефіцієнти матриці ваг випадковим чином і використавши еволюційний алгоритм, отримано навчену нейромрежу.

Коли буде сформована навчальна вибірка і теоретичні знання, можемо починати навчання моделі. Однак проблема полягає в тому, що часто елементи множин представлені в нерівних пропорціях. Ймовірність, що файл є ураженим може бути 5%, а неураженим - 95%. В такому випадку провести навчання неможливе.

Метрика точності в такому випадку повинна бути іншою, та й навчати потрібно теж розумно, щоб нейромережа не зробила такого ж очевидно невірного висновку. Для цього пропонується навчати нейромережу навчальними прикладами, що містять однакову кількість елементів різних класів.

Наприклад, якщо є 20,000 фалйів і з них 1,000 уражених, можна випадковим чином вибрати з кожної групи по 500 прикладів і використовувати

їх для навчання. І повторювати цю операцію раз за разом. Це трохи ускладнює реалізацію процесу навчання, але зате допомагає отримати коректну модель.

Вибравши модель і алгоритм навчання, бажано розділити вибірку на частини: провести навчання на навчальній вибірці, що становить 70% від всієї, і пожертвувати 30% на тестову вибірку, яка буде потрібно для аналізу якості отриманої моделі.

Розробивши алгоритм роботи нейронної мережі потрібно перейти до розробки еврестичного методу на основі сигнатур.

2.2 Сутність еврестичного методу на основі сигнатур

Сучасні дослідження показують, що антивірусні програми відіграють важливу роль в захисті операційних систем від вірусів, та користувачів від витоку інформації, що може призвести до великих втрат на підприємствах, а саме:

– фінансові втрати;

– втрата репутації;

– втрата робочих кадрів і т.д.

Але ці антивірусні програми не є досконалими та потребують допрацювань, алгоритми мають низку недоліків через які шкідливе програмне забезпечення втручається в роботу файлів і ніхто їм не може завадити.

Тому було прийнято рішення після аналізу методів пошуку вірусів, покращити ці методи, а саме:

– еврестичний метод;

– аналіз на основі сигнатур.

Оскільки вони мають низку недоліків і при спробі об'єднання цих методів вони усунуть ці недоліки.

Суть алгоритму заснована на методі пошуку вірусів за допомогою сигнатур але з різним ступенем кореляції. Алгоритм на основі сигнатур вважає файл ураженим, якщо сигнатура співпадає з кодом вірусу на 100%, але це не

надійний спосіб, оскільки файл може бути ураженим вірусом але співпадати з сигнатурою на 90%. Якщо зменшити ступінь кореляції, то покращиться пошук вірусів в файлах.

Після надходження вірусу в операційну систему, алгоритм пошуку вірусів буде перевіряти збіг сигнатури та коду файлу, зчитуючи перший сегмент сигнатури та порівнюючи його з підозрюваним файлом, після порівняння зчитується наступний сегмент сигнатури і так до тих пір доки сигнатура не завершиться. Після перевірки всього файлу визначається чи допустимий рівень співпадінь. Якщо ступень допустимих співпадінь не буде перевищений, то файл вважається не ураженим, а якщо кількість співпадінь перевищений, файл буде вважатись ураженим.

Метод не є надійним на 100%, оскільки в нього є недолік помилкового спрацювання, тому він потребує допрацювань.

Загальна схема пошуку вірусів наведена на рисунку 2.1

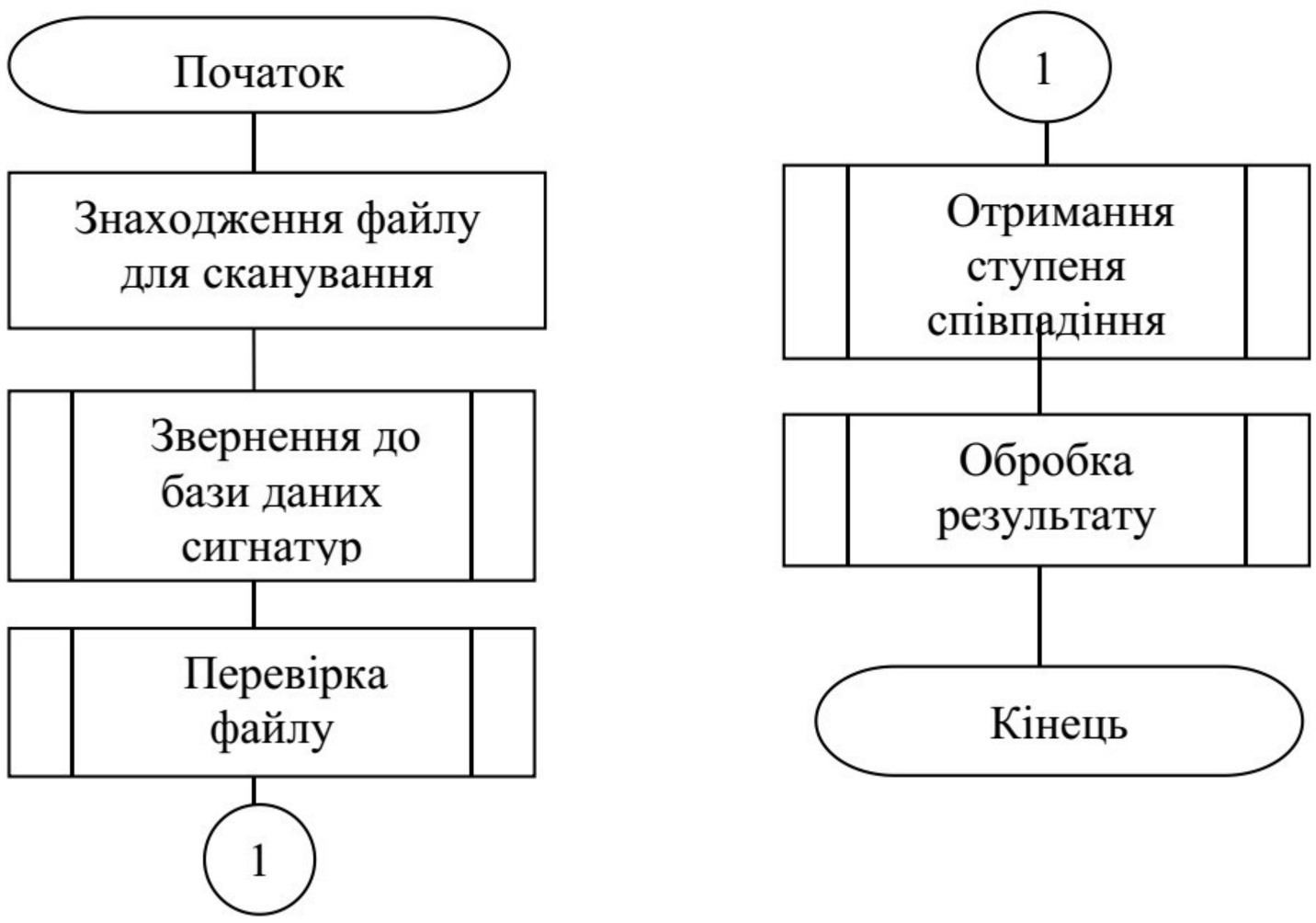


Рисунок 2.1 – Загальна схема пошуку вірусів

Отже, зі схеми стає зрозуміло, як працюватиме алгоритм програмного засобу та як саме він буде усувати недоліки евристичного методу та сигнатурного.

2.3 Алгоритм пошуку шкідливого програмного забезпечення

Програмний засіб використовуватиме власний алгоритм пошуку вірусів.

Перед розробкою програмного засобу потрібно визначитись з алгоритмом роботи програми.

Щоб розпочати пошук шкідливого програмного забезпечення на комп’ютері користувач повинен обрати теку з файлами, які в подальшому будуть проходити перевірку на наявність шкідливого програмного коду. Всі назви файлів будуть зберігатись в програмному засобі, та по черзі кожен файл буде зчитуватись для перевірки.

Отримавши теку з файлами програмний засіб використовуватиме попередньо навчену нейронну мережу для прогнозування загрози, якщо файл вважається ураженим шкідливим програмним забезпечення з точки зору аналізу нейронної мережі то файл запам’ятовується в додатку, для повторної перевірки евристичним методом на основі сигнатур.

Порівняння здійснюватиметься за власним алгоритмом пошуку вірусів. Кожен файл зчитуватиметься по байту, а алгоритм шукає співпадіння з сигнатурою вірусу, якщо співпадіння відбудеться, це співпадіння запам'ятовуватиметься та аналіз буде продовжуватися.

Після перевірки всього файлу, визначатиметься чи кількість співпадіння є допустимою нормою для файлу за формулою(вираз 2.2)

$$R = \frac{N*100}{K},$$

(2.2)

де R – відсоток співпадіння сигнатури шкідливого програмного забезпечення з файлом, N – кількість елементів, які співпали, К – кількість елементів в сигнатурі.

Якщо відсоток вирахуваний перевищує відсоток, який встановлений користувачем, співпадіння сигнатури з файлом, то файл вважатиметься

ураженим та буде потребувати видалення, лікування або занесення в групу карантину.

Отримавши результат перевірки програмний засіб може перейти до його обробки, програмний засіб буде повідомляти після аналізу теки, які саме файли уражені, виводячи назву файлу в елемент керування список, який буде знаходитьсь в програмному засобі на головному вікні.

Також в головному вікні виводитеметься кількість всіх уражених файлів, тека яка аналізувалась, поле статусу, яке показує статус програмного засобу, та файл який проходить процес аналізу.

Після завершення аналізу програмний засіб не буде видаляти файл, не буде заносить в карантин та не буде лікувати, далі користувач сам приймає рішення, що йому робити з ураженим файлом, так як в нього є всі назви уражених файлів.

Метою програмного засобу є аналіз операційної системи та усунення недоліків еврестичного методу на основі сигнатур, а саме:

- ресурсозатратність пошуку;
- прискорення пошуку шкідливого програмного забезпеченя.

Для реалізації програмного засобу потрібно виконати такі задачі:

- аналіз кожного з блоків наведених на рисунку 3.1;
- розробка нейромережі;
- вибір мови програмування;
- розробка загальної схеми роботи програми;
- підготовка бази сигнатур;
- розробка програмного засобу;
- підготовка інформаційного наповнення;
- розробка інтерфейсу програмного засобу;
- розробка вікна налаштувань програмного засобу.

Після аналізу кожного з блоків наведених на рисунку 3.1, можна перейти до обґрунтування вибору програмних засобів для розробки програмного засобу.

2.4 Обгрунтування вибору програмних засобів

Реалізація поставленої задачі, а саме розробка програмного засобу для пошуку вірусів буде проводитися за допомогою мови програмування C#, в програмному середовищі Microsoft Visual Studio.

C# розроблялась як мова програмування прикладного рівня для CLR і тому вона залежить, перш за все, від можливостей самої CLR. Це стосується, перш за все, системи типів C#.

Мова має статичну типізацію, підтримує поліморфізм, перевантаження операторів (в тому числі операторів явного і неявного приведення типу), делегати, атрибути, події, властивості, узагальнені типи і методи, ітератори, анонімні функції з підтримкою замикань, LINQ, виключення, коментарі у форматі XML.

Мова C# – об'єктно-орієнтована мова програмування, що надає можливість повторного використання коду і таким чином економити час на розробку, програми написані по принципах об'єктно-орієнтованого програмування добре структуровані, що дозволяє добре розуміти, які функції виконують окремі підпрограми, такі програми легко модифікувати. Використання об'єктна орієнтованого програмування дозволяє створювати класи – абстрактні типи даних, які визначаються розробником [16].

Проаналізувавши основні особливості мови програмування C#, сформульовано найбільш помітні переваги мови програмування.

Принципово важливою відмінністю від попередників є початкова орієнтація на безпеку коду (що особливо помітно в порівнянні з мовами C і C++) [14].

Уніфікована, максимально близька за масштабом і гнучкості до Common Type System, прийнятої в Microsoft .NET, система типізації є важливою перевагою мови C# [17].

Об'єднання кращих ідей сучасних мов програмування (Java, C++, Visual Basic та ін.) Робить мову C# не просто сумою їх гідністей, а мовою програмування нового покоління[18].

Оскільки дуже важливу роль відіграють розмір, швидкість виконання, наявність дружнього графічного інтерфейсу буде доцільно обрати саме цю мову програмування для реалізації поставленої задачі.

Обравши мову програмування потрібно перейти до розробки загальної схеми роботи програми.

2.5 Розробка загальної схеми роботи програми

Для того щоб перейти до розробки программного засобу потрібно розробити схему роботи додатку.

Загальна схема роботи програми наведена на рисунку 2.2.

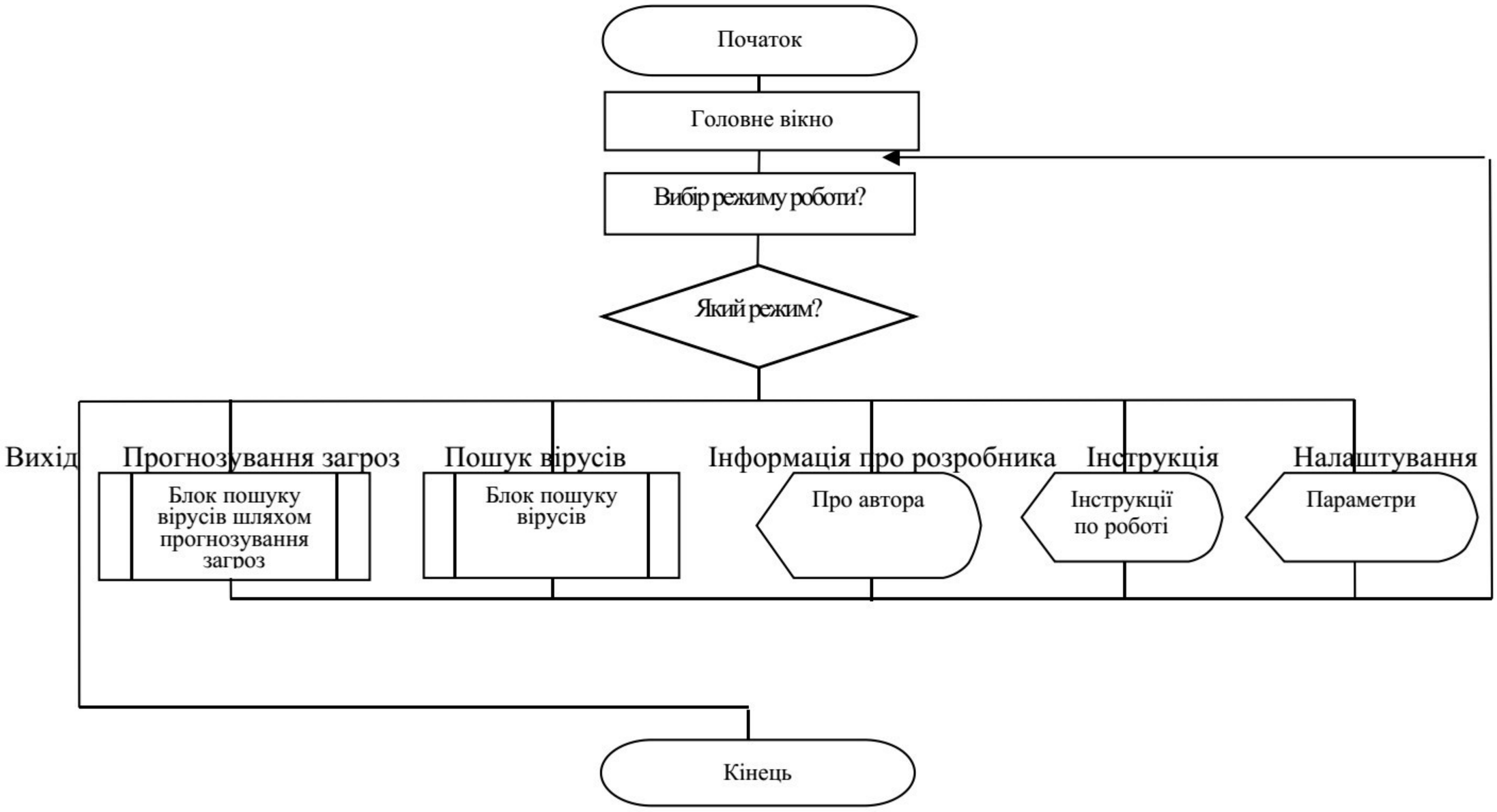


Рисунок 2.2 – Схема функціонування програми

У головному вікні розташовано, елементи керування, які дозволяють керувати роботою програми, а саме:

– передивлятись інформацію про автора;

– читати інструкції по роботі з програмою;
– або безпосередньо перейти до процесу пошуку вірусів на операційній системі.

Отже, із загальної схеми стає зрозуміло, як працює програмний засіб та які режими роботи є. Розробивши загальну схему, можна перейти до підготовки бази сигнатур.

2.6 Розробка бази сигнатур

Базою сигнатур – це сукупність сигнатур шкідливого програмного забезпечення.

Базою сигнатур являється тека в якій зберігаються всі сигнатури, кожна сигнатура зберігається в форматі .txt.

Для створення бази сигнатур було використано декілька тестових вірусів, які використовуються для перевірки антивірусних засобів і не несуть ніяких шкідливих дій (рис 2.3).

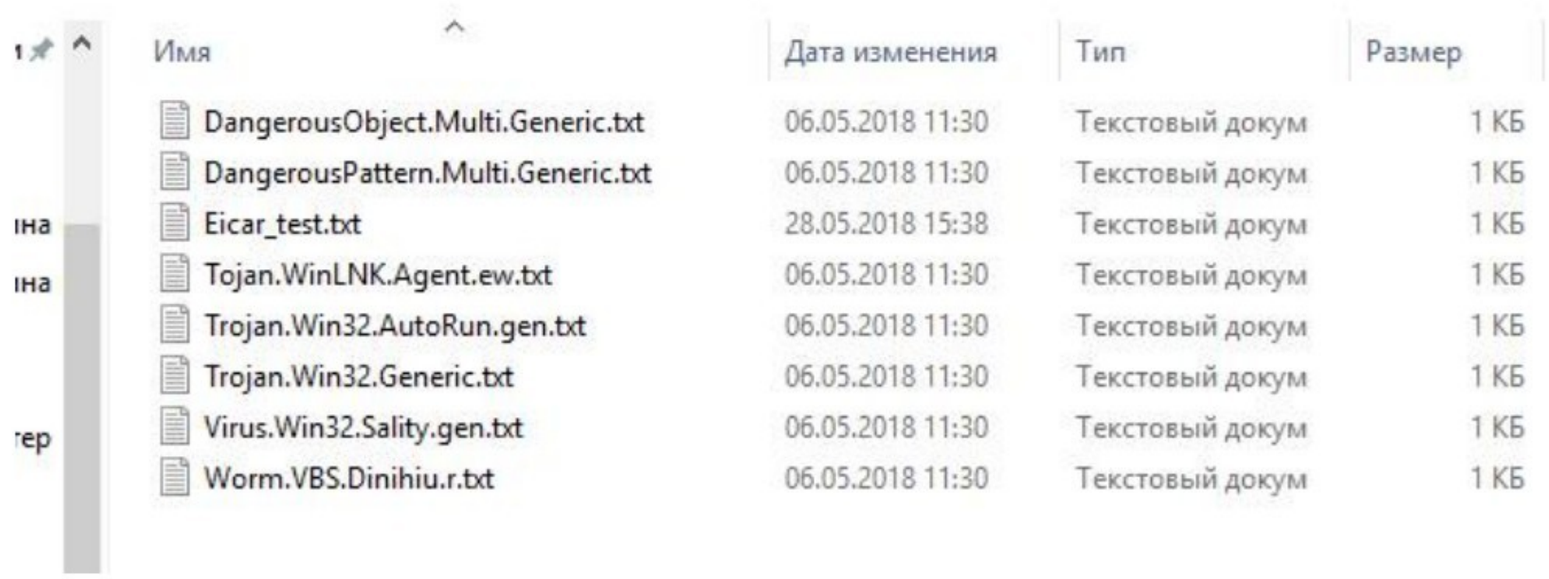


Рисунок 2.3 – База сигнатур

Кожна сигнатура має доволі мале тіло для зменшення кількості хибних спрацювань та зменшення бази сигнатур, так як різні шкідливі програмні засоби можуть використовувати за основу однакові сигнатури (рис.2.4).

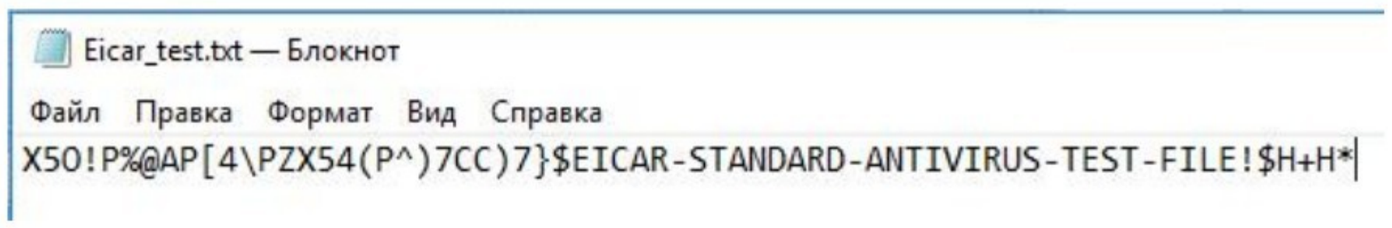


Рисунок 2.4 – Сигнатура шкідливого програмного забезпечення

Розроблена база сигнатур складається тільки з тестових вірусів але в подальшому може складатись з справжніх сигнатур шкідливого програмного забезпечення.

За рахунок того що, що сигнатури зберігаються в .txt форматі користувач може додати власну сигнатуру, як в ручну так і за допомогою програмного засобу, який реалізує можливість поповнення бази сигнатур.

Розробивши базу сигнатур потрібно перейти до реалізації нейронної мережі.

2.7 Реалізаія нейромережі

Для початку потрібно розробити ієрархію класів нейронної мережі. Загальна ієрархія класів нейромережі наведена на рисунку 2.5.

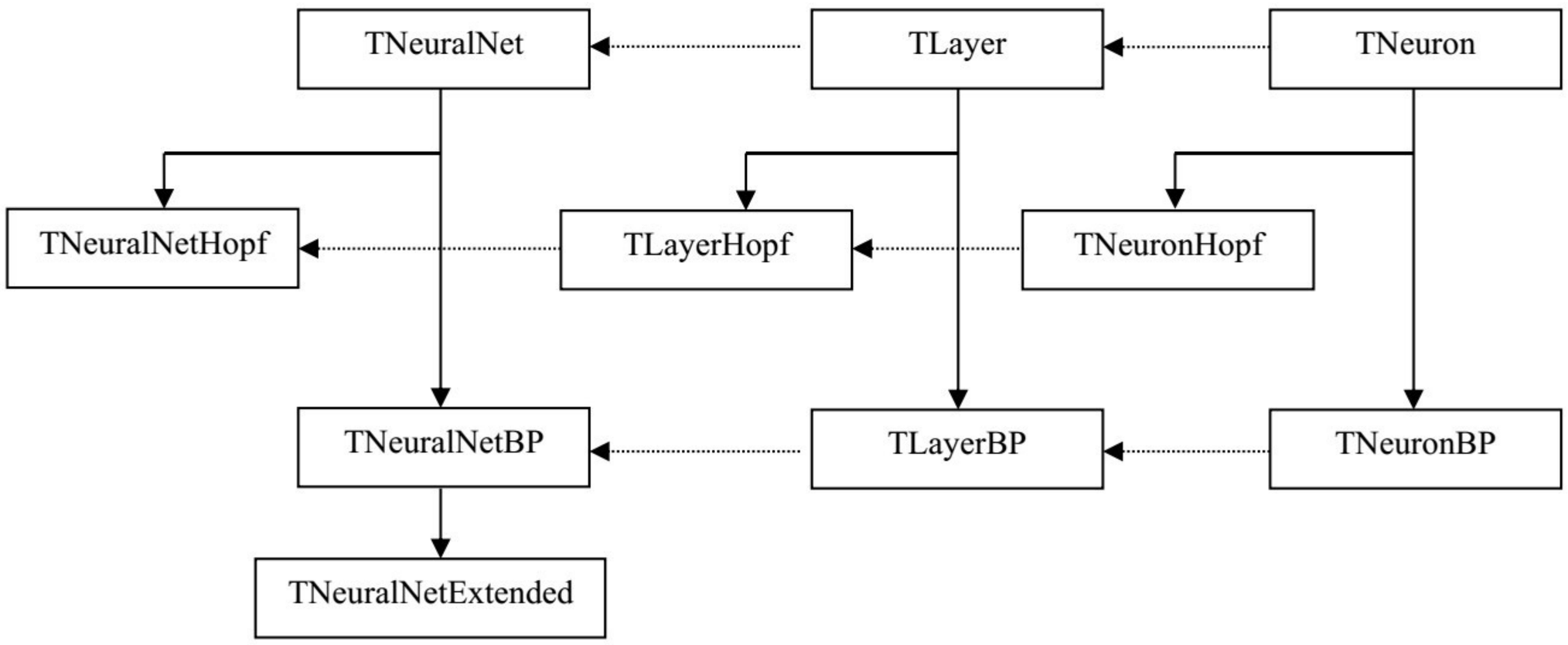


Рисунок 2.5 – Ієрархія класів

Існує три базових класу TNeuron, TLayer, TNeuralNet. Всі інші є похідними від них. На рис. 2.5 приведена ієрархія класів, суцільними лініями

показано успадкування, стрілкою вказаний нащадок, пунктирними в яких класах вони використовуються.

TNeuron є базовим класом для нейронів, несе всю основну функціональність, має індексовану властивість Weights, що представляє собою вагові коефіцієнти, властивість Output, яке є виходом нейрона, тобто результатом обчислень і акумулятор, роль якого, виконує метод ComputeOut.

TNeuronHopf, нащадок TNeuron, реалізує нейрон використовуваний в нейронній мережі Перцептрон, єдиною відмінністю від базового класу, є використання активаційної функції в перекритому методі ComputeOut.

Наступним дочірнім класом, є TNeuronBP служить для програмної реалізації багатошарових нейронних мереж. Абревіатура BP вказує, що нейрон цього типу навчається за алгоритмом зворотного поширення.

Крім того, додані дві властивості, Delta - містить локальну помилку і індексовані властивість PrevUpdate - містить величину корекції вагових коефіцієнтів на попередньому кроці навчання мережі.

Основним призначенням базового класу TLayer і його нащадків TLayerHopf і TLayerBP є об'єднання нейронів в шар, для спрощення роботи з нейронами.

Компонент TNeuralNet базовий компонент для всіх видів нейронних мереж. TNeuralNet забезпечує необхідну функціональність похідних компонентів. Цей компонент підтримує методи для роботи з шарами мережі AddLayer, DeleteLayer і методи для маніпуляцій з вихідними даними AddPattern, DeletePattern, ResetPatterns. Метод Init служить для побудови нейронної мережі. Більшість методів оголошених в розділі public в базовому компоненті і його нащадків - віртуальні, що дозволяє легко перекривати їх.

Компонент TNeuralNetHopf реалізує нейронну мережу Перцептрон.

Додатково включені наступні методи: InitWeights - запам'ятовує пред'явлені зразки в матриці образів і метод Calc - обчислює вихід мережі Перцептрон.

Компонент TNeuralNetBP реалізує багатошарову нейронну мережу, яка навчається за алгоритмом зворотного поширення помилки.

Додатково включені наступні методи: Compute - обчислює вихід нейронної мережі, використовується після навчання мережі; TeachOffLine - навчає нейронну мережу. Компонент дозволяє в режимі design-time, у вікні Object Inspector, конструювати нейронну мережу додаючи або видаляючи шари і нейрони в мережі. Для цього використовується редактор властивостей NeuronsInLayer.

Реалізувавши алгоритм нейронної мережі потрібно перейти до реалізації пошуку вірусів.

2.8 Реалізаія пошуку вірусів

Якщо відсоток співпадінь буде не достатній, щоб можна було стверджувати, що файл є ураженим, то файл буде вважатись не ураженим шкідливим програмним кодом який знаходиться в сигнатурі, з якою відбувалось порівняння (рис 2.6).

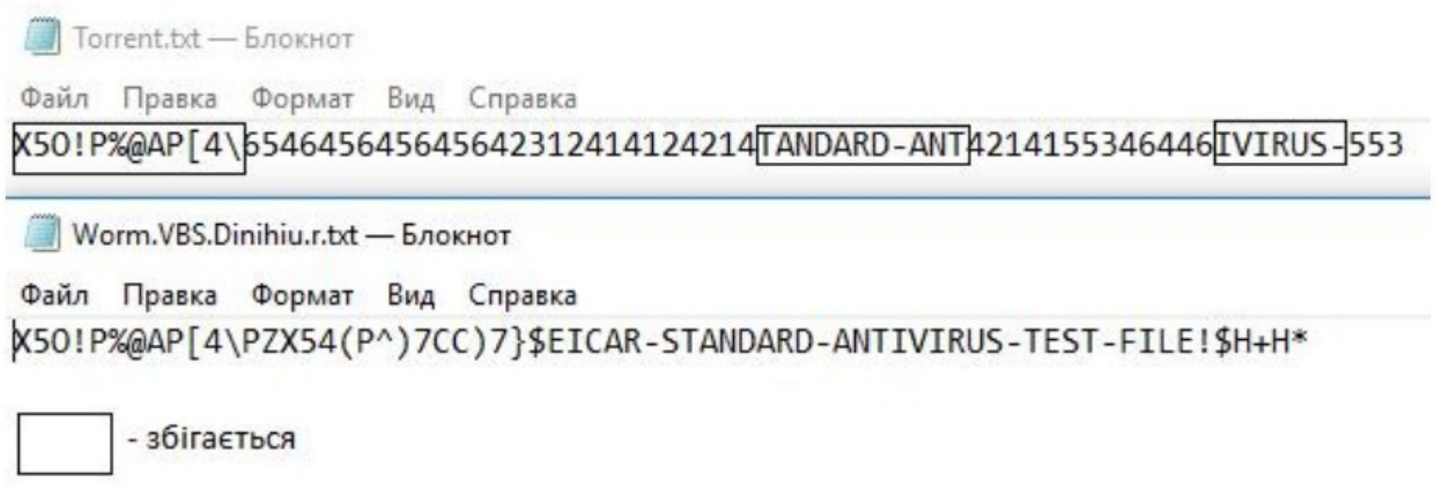


Рисунок 2.6 – Недостатній збіг сигнатури з файлом

Сигнатура складається з 68 байт, кількість співпадінь становить 30 байтів, відповідно кількість співпадінь становить 44%, за формулою(вираз 2.3).

$$\frac{30*100}{68} = 44\%. \quad (2.3)$$

Якщо відсоток співпадінь є не достатнім, то з бази сигнатур зчитується інша сигнатура, яка за аналогічним алгоритмом порівнюється з файлом.

Порівняння буде здійснюватись до тих пір поки всі сигнатури не пройдуть процес порівняння з файлом або файл не співпаде з сигнатурою на достатній рівень, що буде означати, що файл є ураженим шкідливим програмним кодом.

Також алгоритм має здатність обходити мертвий код, тобто код, який міститься в підозрілому файлі, та ніяк не впливає на виконання файлу (рис. 2.7).

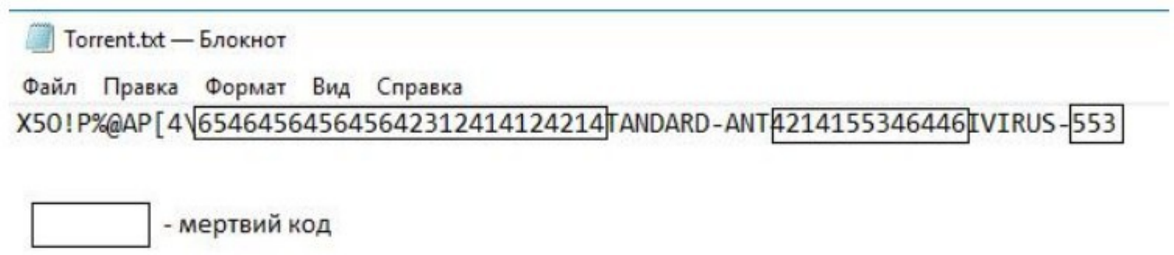


Рисунок 2.7 – Вигляд мертвого коду

Та перехід з одного місця у файлі на інший. За рахунок пошуку «опкоду» команди переходу з одного місця у файлі на інший.

Для того, щоб розпочати пошук вірусів, потрібно вибрати теку для сканування на предмет пошуку уражених файлів. Для вибору теки було створено кнопку "Browse", яка відкриває панель вибору (див. рис. 2.8).

Після вибору теки в поле lblfolder записується назва обраної теки, з'являється поле lblviruses в якому міститься кількість уражених файлів, онулюється змінна count в якій міститься кількість уражених файлів, оновлюється поле статусу та очищується поле списку.

Для того, щоб приступити до процесу пошуку уражених файлів була підготовлена кнопку "Scan", при натисканні на неї створюється колекція та зчитується з файлу в неї сигнатура вірусу.

Зчитавши сигнатуру вірусу йде отримання назв всіх файлів обраної теки, для цього створюється колекція.

Отримавши назви всіх файлів теки, перед порівнянням сигнатури і файлу потрібно зчитати файл, для цього відкривається потік та в буфер зчитується файл.

Далі буде створено 4 функції, які потрібно розглянути.

Схема алгоритму пошуку наведена на рисунку 2.8

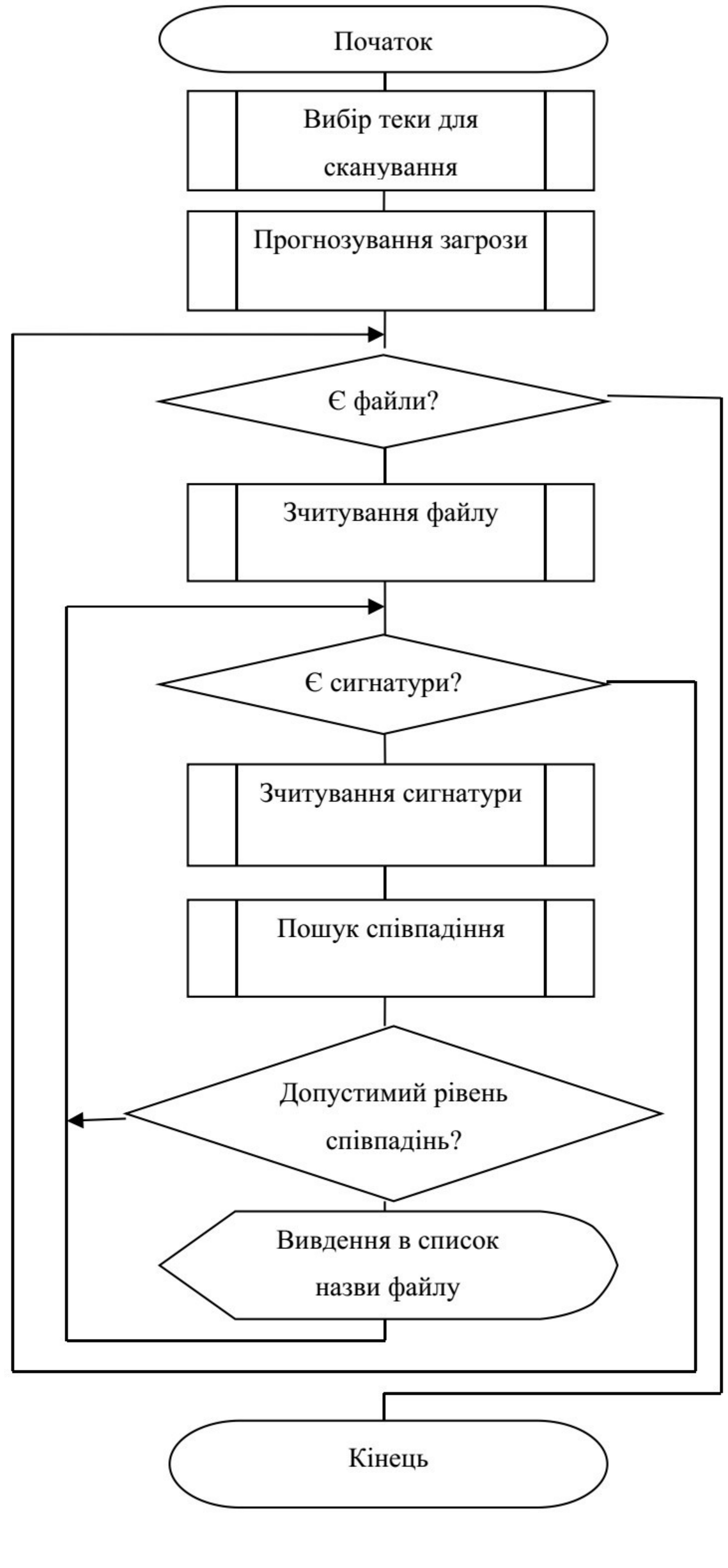


Рисунок 2.8 – Схема алгоритму пошуку шкідливого програмного забезпечення

Отже, розробивши схему пошуку шкідливого програмного забезпечення, можна перейти до розробки основних функцій еврестичного методу пошуку на основі сигнатур.

2.9 Розробка основних функцій евристичного методу пошуку шкідливого программного забезпечення

Головна функція algoritmWithJmp в якій викликаються всі інші функції та яка відповідає за:

– знаходження першого співпадіння сигнатури шкідливого програмного коду та файлу;

– пошук «опкоду» переходу з однієї точки файлу в інший;

– обрахунок відсотку співпадінь сигнатури та файлу.

Функція приймає такі параметри, як byte[] VirusListString, в якому міститься весь файл який перевіряється, параметр byte[] read, в якому міститься сигнатура шкідливого програмного забезпечення, параметр ref float KilkictbVsixPorivnyanb, в якому міститься довжина сигнатури, параметр ref float KilkictbSpivpadinb, в якому міститься кількість співпадінь, параметр float VidsotkoveSpivvidnishenyaSpivpaninya, відсоток співпадінь, List<string> AllFoundViruses, список всіх знайдених вірусів, та патаметр string item шлях до файлу який сканується. Всі ці параметри в подальшому передаються в інші функції.

Прототип функції.

```
public void algoritmWithJmp(byte[] VirusListString, byte[] read, ref float KilkictbVsixPorivnyanb, ref float KilkictbSpivpadinb, float VidsotkoveSpivvidnishenyaSpivpaninya, string item, List<string> AllFoundViruses)
```

Для знаходження співпадіння здійснюється порівняння байту сигнатури та байту файлу, який перевіряється if (read[j] == VirusListString[i]). Якщо співпадіння знайдено викликається наступна функція algoritmWithJmpSecond, для подальших обрахунків, після її виконання обраховується відсоткове співпадіння.

```
VidsotkoveSpivvidnishenyaSpivpaninya = KilkictbSpivpadinb / KilkictbVsixPorivnyanb;
```

Обрахувавши співпадіння викликається функція virusCheck для подальшого оброблення результату.

virusCheck(VidsotkoveSpivvidnishenyaSpivpaninya, ref count, xRoot, item, AllFoundViruses, endIndexforListBox).

Для того щоб дізнатись коли саме відбувається перехід з одного місця на інше у файлі було використано «опкод» команди jmp, його значення становить 233, 234 або 235 залежно від параметрів з якими він працює,

```
else if (read[j] == 233 || read[j] == 234 || read[j] == 235){
```

Якщо «опкод» був знайдений, алгоритм програми спробує знайти куди саме відбувся перехід викликаючи функцію algoritmSearchJmp.

Функція підрахунку кількості послідовних співпадінь

Наступна функція algoritmWithJmpSecond підраховує кількість послідовних співпадінь, враховуючи перехід з однієї точки файлу в інший та повертає значення для подальших обрахунків.

Прототип функції.

algoritmWithJmpSecond(VirusListString, read, i, j, ref KilkictbSpivpadinb);

Для знаходження співпадіння відбувається порівняння сигнатури та файлу який перевіряється на наявність шкідливого програмного забезпечення, та якщо вони співпадають, співпадіння запам’ятовується та порівнюються наступні елементи сигнатури та файлу.

```
if (VirusListString[i] == read[j]){KilkictbSpivpadinb++;
i++;j++;}
```

Якщо співпадіння не відбувається або було знайдено «опкод» команди jmp функція завершує свою роботу викликаючи функцію algoritmSearchJmp, для подальших обрахунків.

Якщо «опкод» не був знайдений та всі байти сигнатури співпали з байтами файлу, який перевіряється, функція повертає результат, в функцію algoritmWithJmp для подальших обрахунків.

Функція пошуку переходу

Функція algoritmSearchJmp відповідає за пошук точки у файлі куди відбувся перехід, та за обхід мертвого коду.

Прототип функції.

algoritmSearchJmp(byte[] VirusListString, byte[] read, int i, int j

Для пошуку точки у файлі, здійснюється пошук співпадіння наступного елементу сигнатури та файлу, якщо співпадіння буде знайдено, функція завершує свою роботу повертаючи порядковий номер байта, де відбулось співпадіння, щоб функція algoritmWithJmpSecond, продовжила свою роботу з точки співпадіння сигнатури та файлу який перевіряється.

```
for (; j <read.Length;){
if (VirusListString[i] == read[j]){return j;}
if (j + 1 == read.Length || i + 1 == VirusListString.Length) return j; j++;}
return j;
```

Якщо співпадіння не було знайдено функція завершує свою роботу.

Функція обробки результату

Функція virusCheck відповідає за виведення назви файлу на екран, якщо він являється ураженим шкідливим програмним кодом.

Щоб дізнатись чи файл є ураженим шкідливим програмним кодом, відбувається порівняння відсоткового співпадіння з допустимим рівнем співпадінь.

```
if (VidsotkoveSpivvidnishenyaSpivpaninya > rate)
```

Якщо умова виконується, в головне вікно програмного засобу виводиться назва файлу без шляху до нього, так як шлях може бути занадто великим, було прийнято рішення виводити тільки назву файлу. Для того, щоб виводилась тільки назва файлу від item відкидаються всі непотрібні дані та виводяться на головне вікно програмного засобу.

```
int endIndexListBox = item.Length;
Program.Form.listBox1.Items.Add(item.Substring(endIndexforListBox + 1,
endIndexListBox - endIndexforListBox - 1));
```

Після того, як файл було виведено на головне вікно програмного засобу, всі операції з цим файлом завершуються, та зчитується новий файл для перевірки.

Розробивши програмний засіб потрібно підготувати інформаційне наповнення.

2.10 Підготовка інформаційного наповнення

Інформаційним наповненням даної програми є графічні ресурси, аудіо файли та файли формату xml в яких міститься переклад для інтерфейсу.

Було прийнято рішення графічні ресурси імпортувати в програмний засіб (рис. 2.9).

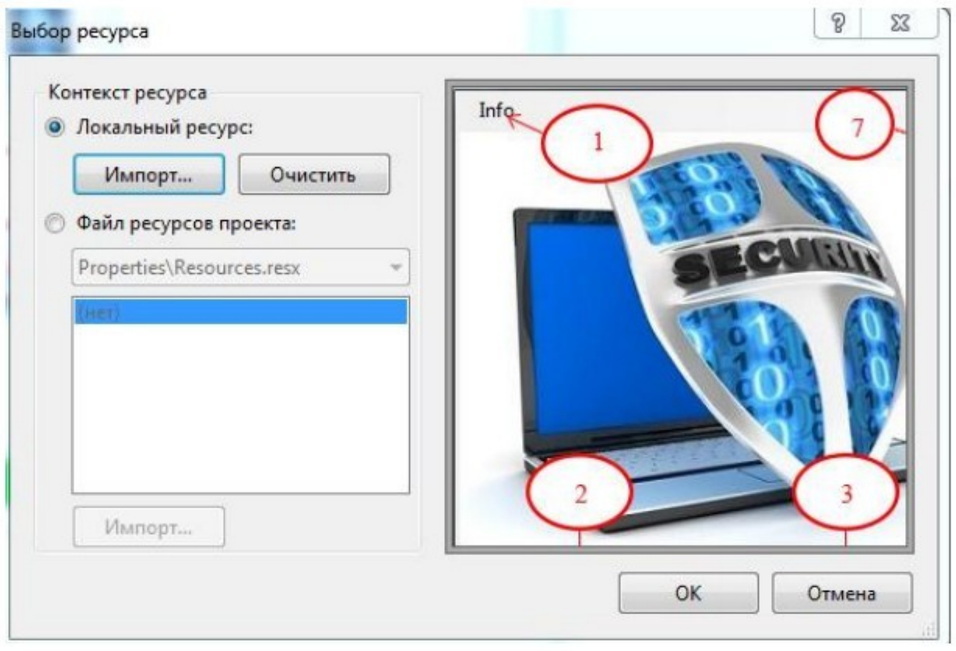


Рисунок 2.9 – Вигляд вікна імпортування графічного ресурсу

Аудіо ресурси та ресурси формату xml, містяться в одній теці (рис. 2.10).

Файли формату xml дають змогу користувачеві змінити мову інтерфейсу програмного засобу, в програмному засобі присутні 2 мови українська та англійська. Аудіо ресурси виступають у кількості одного аудіо файлу для попередження користувача про те, що програмний засіб виявив шкідливе програмне забезпечення.

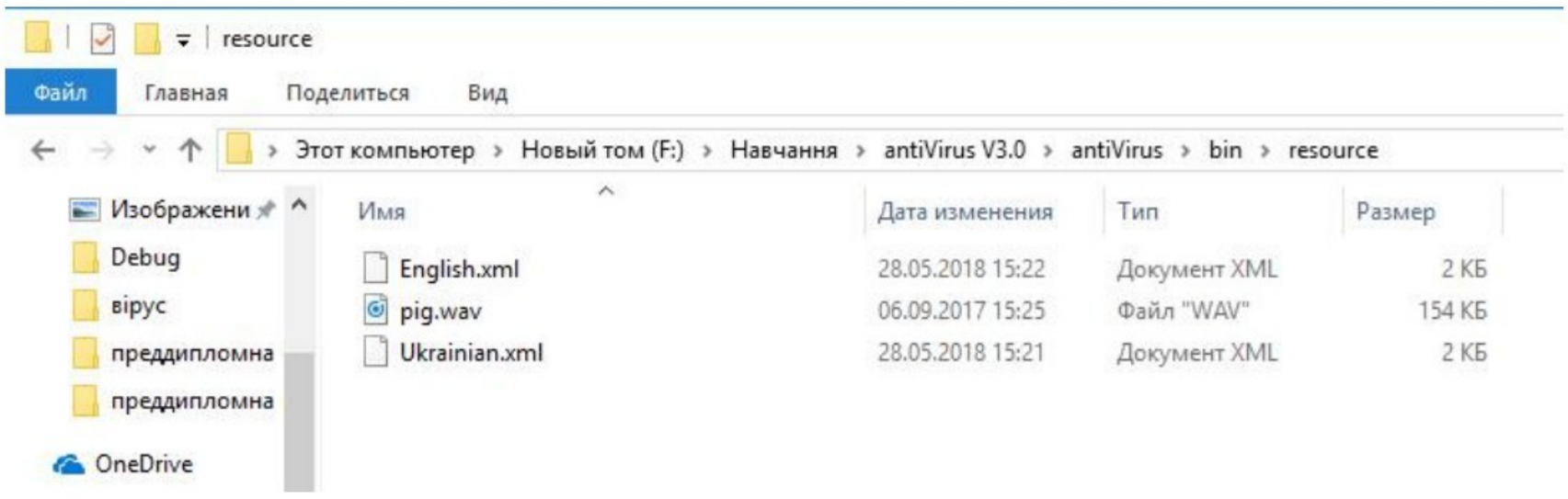


Рисунок 2.10 – Вигляд теки з ресурсами

Графічні ресурси містяться на головному вікні програмного засобу та в додаткових вікнах в яких знаходиться інструкція і інформація про автора (див.рис. 2.11).

Схема ресурсів наведена на рисунку 2.11.

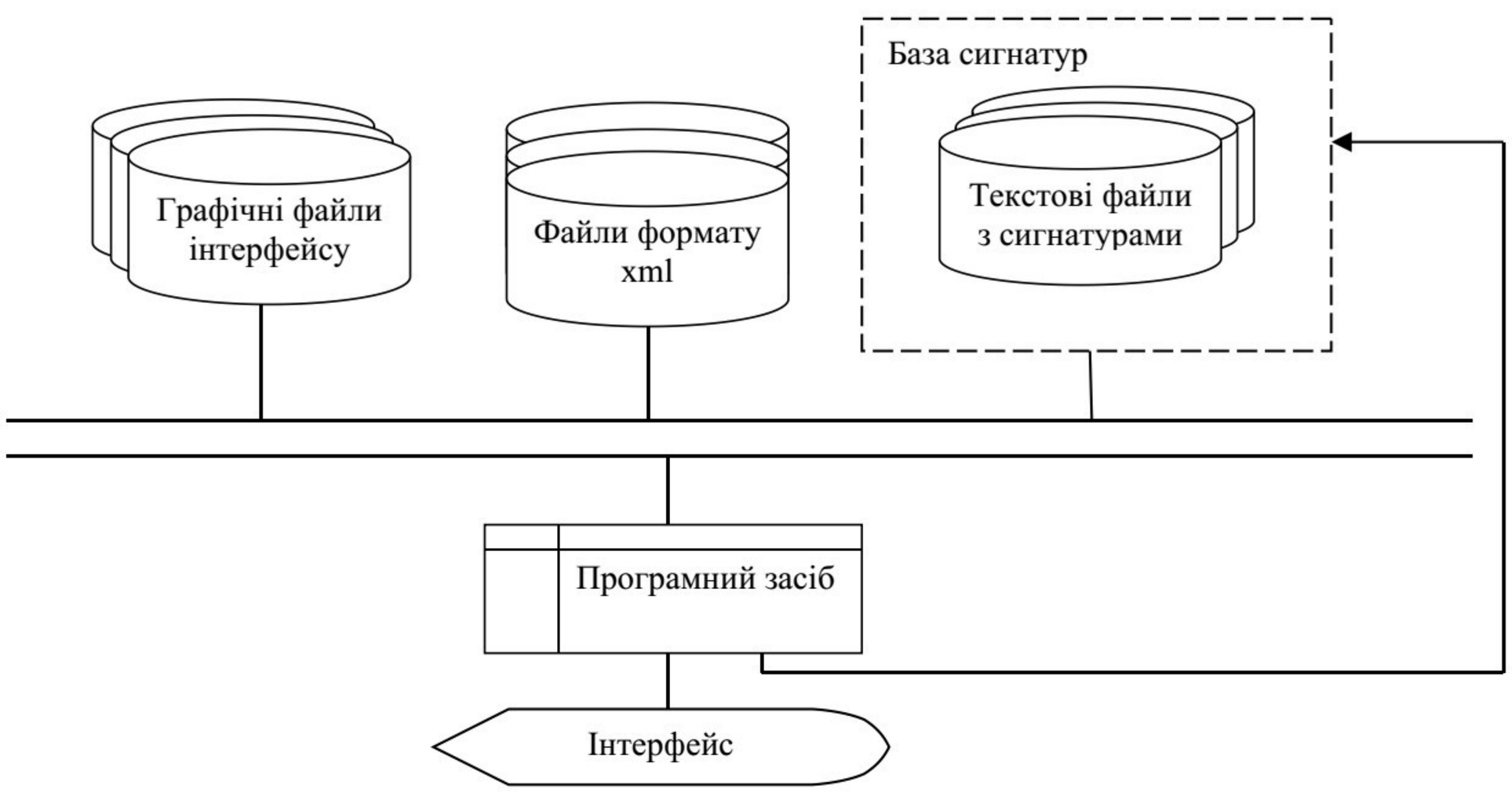


Рисунок 2.11 – Схема ресурсів програми

Розробивши схему ресурсів програми, стає зрозуміло, які ресурси взаємодіють з програмою та можна перейти до створення інтерфейсу програмного засобу.

2.11 Розробка інтерфейсу програми

Головною частиною програми є процес пошуку вірусів, тому слід реалізувати зручний інтерфейс та надати програмі естетичного вигляду.

Для надання естетичного вигляду програмному засобу було прийнято встановити на головному вікні зображення, яке додасть додатку кращого вигляду.(рис 2.12).

Рисунок 2.12 – Вигляд зображення на головному вікні

Для зручності користувача було вирішено реалізувати пошук вірусів в одному вікні (рис 2.13).

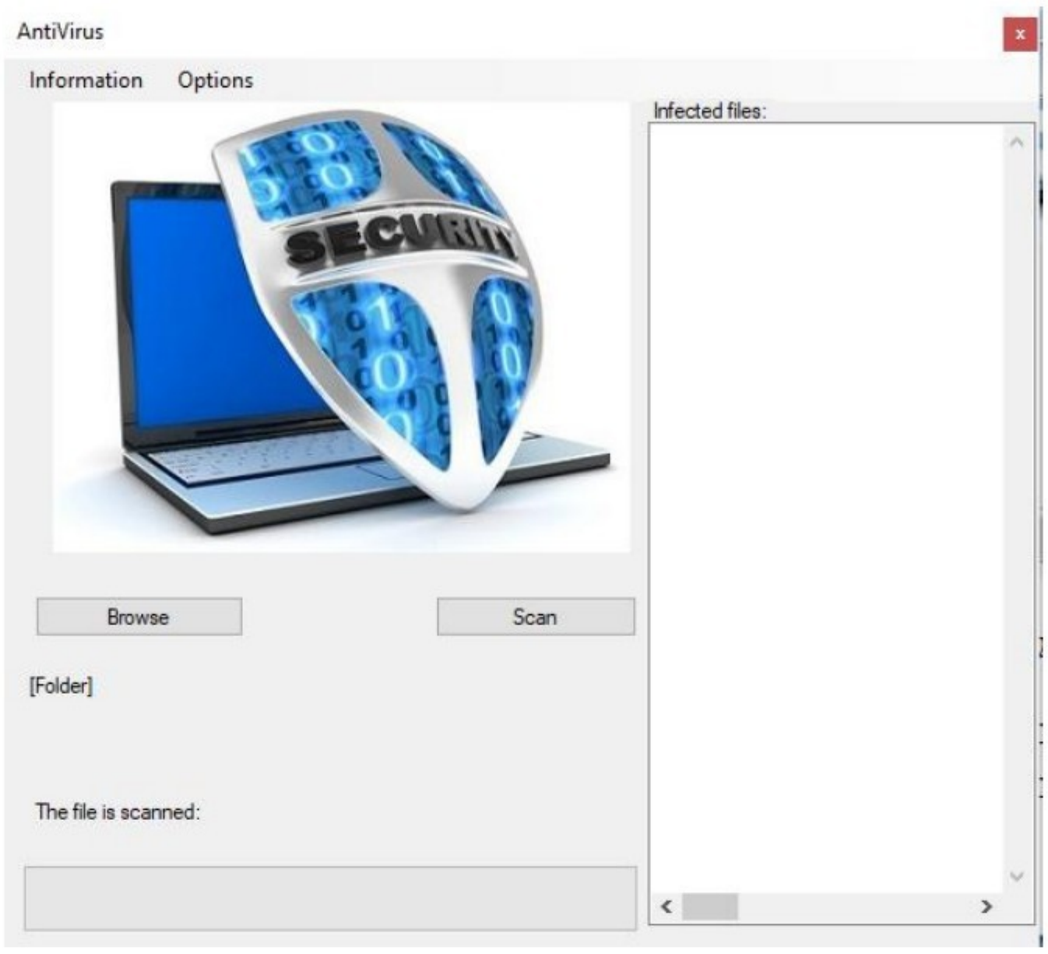


Рисунок 2.13 – Вигляд основного вікна додатку

Було створено вікно в якому користувач може змінювати параметри програмного засобу, а саме:

– вибір мови інтерфейсу програмного засобу;

– включення аудіо ресурсів;

– вибір формату файлів, як будуть в подальшому перевірятись на наявність шкідливого програмного коду;

– встановити відсоток співпадінь сигнатури вірусу з файлом який перевіряється при якому файл буде вважатись ураженим шкідливим програмним кодом;

– додати свою сигнатуру яка буде використовуватись програмним засобом для пошуку шкідливого програмного коду (рис. 2.14).

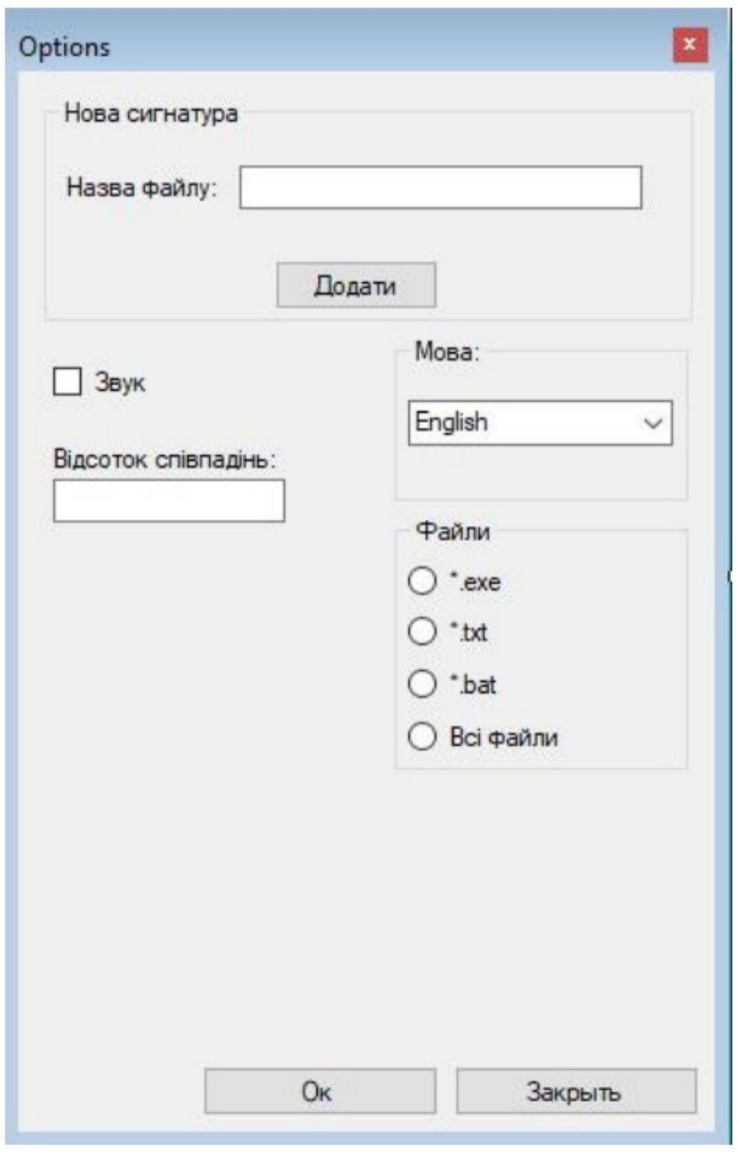


Рисунок 2.14 – Вигляд параметрів програмного засобу

На головному вікні програмного засобу (див. рис. 2.15) зображені елементи керування, вони додані для зручності користувача та керуванням програмним засобом.

За допомогою кнопки "Browse" можна вибрати теку, в якій потрібно здійснити перевірку на віруси (рис. 2.15).

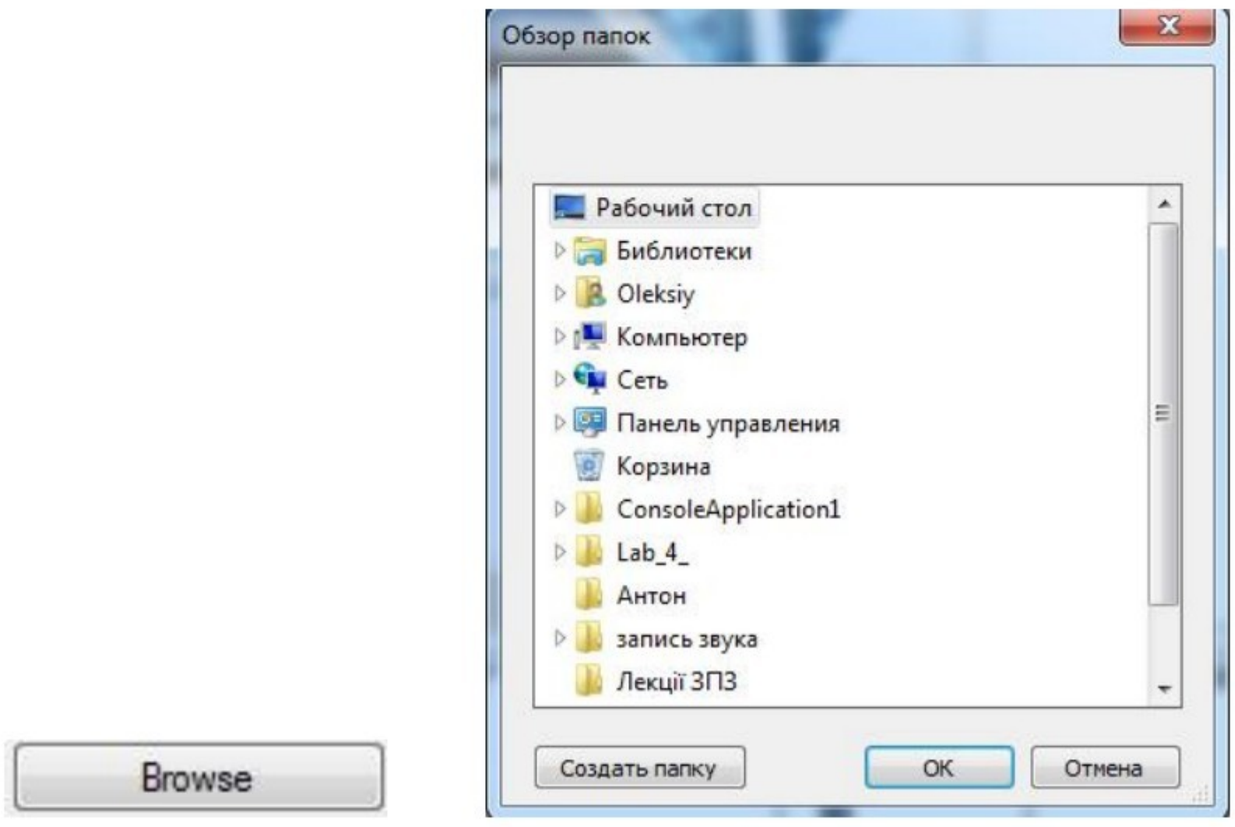


Рисунок 2.15 – Вигляд вибору теки

За допомгою кнопки "Scan", можна запустити процес пошуку вірусів.

Текстові поля, які розташовані на головному вікні, програмного засобу, призначенні для надання користувачу такої інформації:

– тека, яка сканується;
– кількість знайдених вірусів;
– файл, який проходить процес сканування (рис. 2.16).

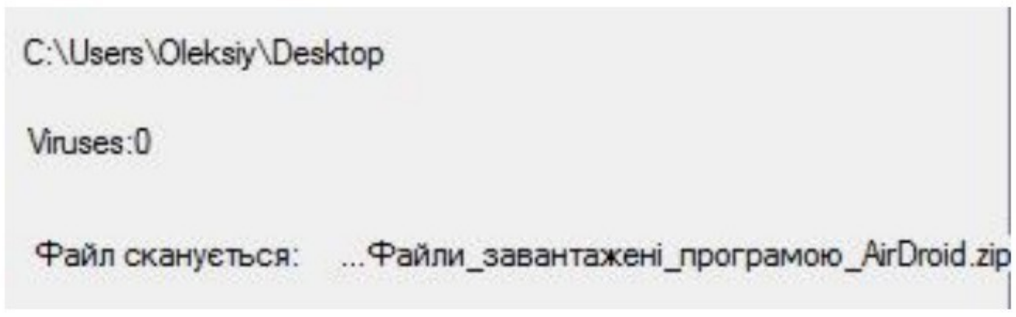


Рисунок 2.16 – Вигляд текстових полів

Список призначений для виведення назви та шляху до всіх уражених файлів (рис 2.17).

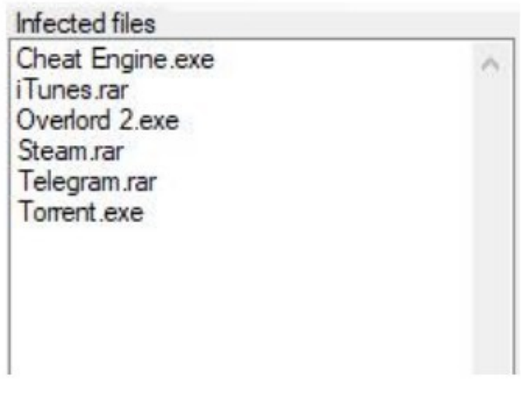


Рисунок 2.17 – Вигляд списку уражених файлів

Для того, щоб було зрозуміло на якому етапі сканування знаходиться програмний засіб, було прийнято рішення додати поле статусу.

Отже, розробивши інтерфейс програмного засобу, можна перейти до створення параметрів налаштування програмного засобу.

2.12 Налаштування програмного засобу

Кожен антивірусний засіб містить ряд параметрів, які користувач може налаштувати, це може бути як змінна візуального вигляду антивірусного засобу, так і налаштування пошуку шкідливого програмного забезпечення або його відключення. Це виконано для полегшення взаємодії користувача з антивірусним засобом, щоб кожен користувач не залежно від того чи він є

новачком чи досвідченим користувачем антивірусних засобів, міг налаштувати програмний засіб так, як йому завгодно.

Тому було прийнято рішення під час розробки програмного засобу створити низку параметрів для користувача (див. рис. 2.14). Майже кожен з параметрів відіграє важливу роль в пошуку шкідливого програмного коду у файлах. Налаштовуючи кожен з них користувач може, як покращити пошук шкідливого програмного коду, так і його погіршити.

Однією з функцій програмного засобу є надання користувачеві можливості додати до бази сигнатур, власну сигнатуру, яка буде в подальшому використовуватись під час пошуку шкідливого програмного коду у файлах (рис.2.18).

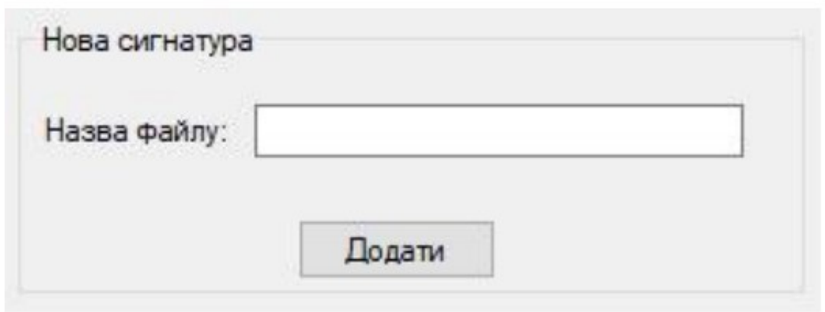


Рисунок 2.18 – Додання сигнатури

Щоб додати сигнатуру користувач повинен ввести назву файлу в якій буде зберігатись сигнатура та натиснути кнопку «Додати» після чого відкриється створений текстовий файл.

```
System.IO.FileStream fs = System.IO.File.Create(textbox + ".txt");
fs.Close();
Process.Start(textbox + ".txt");
```

Після того, як користувач запише в текстовий файл сигнатуру шкідливого програмного забезпечення, текстовий файл буде додано до бази сигнатур (рис. 2.19).

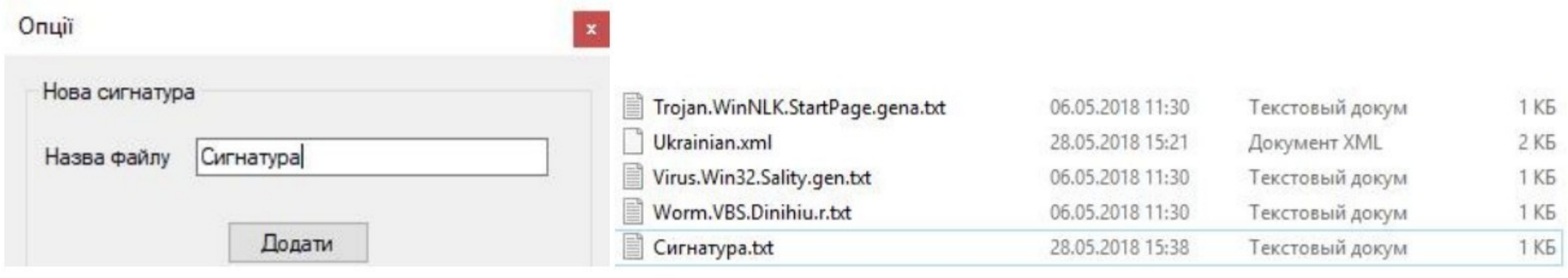


Рисунок 2.19 – Збереження файлу до бази сигнатур

Також користувач може вибрати формат файлів які будуть перевірятись на наявність шкідливого програмного забезпечення (рис. 2.20).

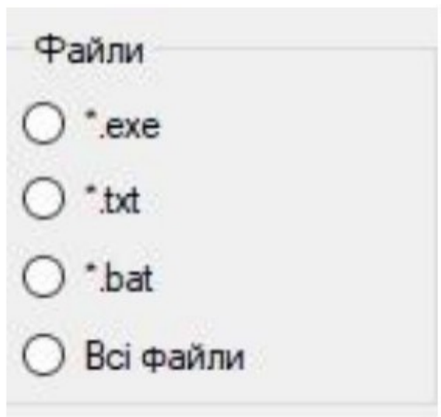


Рисунок 2.20 – Вибір формату файлів

Користувачу надається можливість вибору з чотирьох запропонованих варіантів, а саме: exe, txt, bat або перевірка в усіх запропонованих форматах. Під час процесу пошуку шкідливого програмного забезпечення програмний засіб буде аналізувати тільки ті файли, які вибрав користувач.

```
List<string> search = Directory.GetFiles(directories, TypeOfFiles).ToList();
```

Де TypeOfFiles – це вибраний користувачем формат файлів.

Якщо користувач не розуміє мови яка використовується в інтерфейсі програмного засобу він може її змінити, використовується дві мови: англійська та українська (рис 2.21).

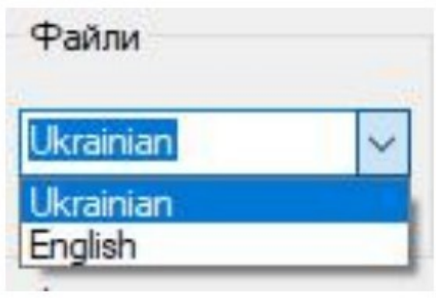


Рисунок 2.21 – Вибір мови

Щоб програмний засіб міг використовувати дві різні мови було використано файли формату xml, для завантаження потрібної мови.

```
        nameXML = comboBox1.SelectedItem.ToString();
        XmlDocument xDoc = new XmlDocument();
        xDoc.Load(Options.nameXML + ".xml");
        XmlElement xRoot;
        xRoot = xDoc.DocumentElement;
        Music = checkBox1.CheckState.ToString();
        Program.Form.Text = func.getAtribut("AntiVirus", xRoot);
        Program.Form.infoToolStripMenuItem.Text =
func.getAtribut("Information", xRoot);
```

```
Program.Form.optionsToolStripMenuItem.Text =
func.getAtribut("Options", xRoot);
```

Для попередження користувача про знаходження шкідливого програмного забезпечення використовується аудіо файл.

```
string path = "pig.wav";
System.Media.SoundPlayer player = new System.Media.SoundPlayer();
player.SoundLocation = path;
player.Load();
player.Play();
```

Програмний засіб вважає файл ураженим шкідливим програмним кодом, коли сигнатура та файл який перевіряється співпадає один одному на певний відсток, який користувач може змінити. За замовчуванням відсоток становить 85.

```
if (!Int32.TryParse(textBox2.Text, out rate)){
MessageBox.Show(unCorrect, error, MessageBoxButtons.OK);return;}
else if (rate > 100 || rate < 0){ MessageBox.Show(unCorrect, error,
MessageBoxButtons.OK);return;}
```

Де, rate – це відсоток співпадінь.

Отже розробивши алгоритми пошуку шкідливого програмного забезпечення, реалізувавши додаток та підготувавши інформаційне наповнення можна перейти до тестування програмного засобу.

3 ЕКСПЕРЕМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

3.1 Вибір тестів для перевірки

До якісних і достовірним тестувань методів пошуку шкідливих програмних забезпечень потрібен дуже серйозний підхід. Фахівці в області комп'ютерної безпеки використовують дороге устаткування і застосовують трудомісткі операції для тестування ефективності антивірусних продуктів.

Тим не менш, деякі компанії пропонують безпечні рішення для користувачів, які дозволяють перевірити захист встановленого антивірусу. Як правило організації пропонують веб-сторінки або завантаження, які містять урізані, неактивні або імітовані версії шкідливих додатків. Будь-який користувач може використовувати такі об'єкти для пошуку вразливих місць в захисті без ризику збитку для системи і даних. Вони, звичайно, не дадуть повну картину, але в будь-якому випадку виявляться корисними. Наприклад, коли на комп’ютері оновлено або змінено основний антивірусний захист, можна провести декілька експрес-тестів, щоб перевірити, чи активовано основні модулі рішення.

Звичайно, дані випробування мають декілька значних обмежень. Так, наприклад, користувачі не можуть переконатися в коректності тестової процедури і правильності імітації діяльності шкідливих програм, а також в релевантності методики тестування по відношенню до шаблонів використання і рівню навичок.

Жоден тест не може враховувати патерни поведінки як користувачів з досвідом, так і новачків. Наприклад, випробування, які аналізують базовий захист, можуть бути достатніми для досвідчених користувачів, але не повноцінними у випадку з початківцями користувачами.

Також необхідно пам'ятати про потенційно приховані дії, які можуть включати в себе антивірусні програми. Наприклад, деякі вендори можуть розробляти тести, які штучно підкреслюють сильні сторони продукту.

За результатами аналізу інформаційних джерел, наведемо перелік безкоштовних і добре відомих тестів, які можуть використовувати для симуляції шкідливої атаки (табл. 3.1) [19].

Таблиця 3.1 – Відомі тести для антивірусних програм

Назва	Розробник тестів	Основні характеристики
Anti-malware Testfile	Європейський інститут комп'ютерних антивірусних досліджень (European Institute for Computer Anti-Virus Research)	В основі – тестовий вірус «Eicar.com»
Test your anti-Malware solution!	WICAR.org	В основі – тестовий вірус «Eicar.com»
Security features check	Організація по стандартам тестування засобів боротьби зі шкідливими програмами (Anti-Malware Testing Standards Organization)	В основі – тестовий вірус «Eicar.com»
SmartScreen	Microsoft	Доступні лише демо-сторінки
Test Your Metal	Fortinet	Працює лише в online-режимі
CheckMe	Check Point	Дозволяє перевірити власний комп’ютер в режимі online, але бази тестів не надає
LeakTest	Gibson Research Corp.	Віддалене тестування
Security Test Tool	SpyShelter	Програмний засіб, який імітує декілька методів, які використовують шкідливе програмне забезпечення, але бази тестів не надає

Антивірусні компанії активно діляться такими базами між собою. Відповідна база з популярного антивіруса ClamAV доступна всім бажаючим. На сайті компанії можна знайти посилання на останню версію антивірусного

продукту, а також посилання для скачування вірусних баз ClamAV [20]. Однак база ClamAV знаходиться у якомусь специфічному форматі і отримати «чистий» код сигнатур з цієї бази наразі не вдалося.

Як бачимо, найбільше інформації про тестовий вірус Eicar, який більшість розробників використовують для перевірки власних антивірусних продуктів.

3.2 Тестовий вірус Eicar для перевірки працездатності методів пошуку

Під іменем EICAR-Test-File детектується невеликий 68-байтовий COM-файл, який не може бути вірусом, а всього лише виводить текстове повідомлення і повертає керування операційною системою.

Деякий час назад розробники декількох антивірусних програм почали включати в свої пакети подібні файли, які не є вірусами, але визначаються антивірусом як вірус. Це викликано інтересом з боку користувачів, які "живих" вірусів не мають, але бажають подивитися, яким чином антивірус реагує на віруси, які повідомлення при цьому виводить і які дії пропонує здійснити користувачу.

Розробники антивірусних програм вирішили виробити єдиний стандарт на подібний "вірус емулятор", а згодом прийшли до висновку, що ця програма повинна складатися тільки з текстових повідомлень. Це необхідне для того, щоб найбільш цікаві користувачі могли набрати її самостійно (наприклад, під диктовку по телефону або переписавши з документації). В результаті цей COM-файл виглядає так:

X5O!P%@AP[4\PZX54(P^)7CC)7)$EIKAP-СТАНДАРТ-АНТИВІРУС-ТЕСТ-ФАЙЛ!$H+H*

При запуску COM-файл виводить повідомлення: «EICAR-STANDARD-ANTIVIRUS-TEST-FILE!» і повертає управління операційній системі. Більшість антивірусних програм детектують його як тестовий файл EICAR [21].

EICAR не перевіряє, наскільки оперативно розробники реагують на віруси і наскільки якісно виліковуються заражені файли – для цього потрібен база сигнатур свіжих вірусів. Його завдання інше: продемонструвати працездатність антивірусної системи і вказати, які об'єкти перевіряються антивірусом, а які – ні.

Подальше тестування проводитиметься з Eicar тест вірусом для перевірки на працездатність програмного засобу.

3.3 Тестування програмного засобу

Перевірка алгоритму пошуку шкідливого програмного забезпечення програмного засобу проводитиметься на 6 варіантів уражених файлів.

підозрілий файл складається на 100% тестового вірусу Eicar;

підозрілий файл складається з тестового вірусу Eicar менш ніж на 100% але більше допустимого рівня;

підозрілий файл містить 100% тестового вірусу Eicar але також містить мертвий код;

підозрілий файл містить менш ніж 100% тестового вірусу Eicar але більше допустимого рівня та містить мертвий код;

підозрілий файл містить 100% з тестового вірусу Eicar, містить мертвий код та містить «опкод» команди jmp, для переходу з одної точки пам’яті файлу в іншу;

підозрілий файл містить менше допустимого рівня, щоб вважати файл ураженим шкідливим програмним кодом.

Метод пошуку шкідливих програмних засобів шляхом прогнозування загроз є ресурсозатратним, тому його доцільно використовувати тільки в тому випадку коли потрібно проаналізувати велику кількість даних, якщо її використовувати для перевірки декількох незначних файлів, то перевірка займе більше часу чим евристичним методом пошуку шкідливого програмного забезпечення, тому було прийнято рішення проводити тестування на теці з

файлами, які займають 25 мегабайт вільного місця на диску, та містять один вірус.

Для тестування створено файл та записано в нього сигнатуру тестового вірусу Eicar.

Після створення файлу було перевірено чи програмний засіб буде вважати цей файл вірусом та зафіксовано час аналізу теки, в обох випадках уражений файл був знайдений (рис 3.1, а).

Також уражений файл може містити не все тіло шкідливого програмного засобу, тому було перевірено, як програмний засіб впорається з ураженим файлом тестовим вірусом менш ніж на 100%, але з достатнім рівнем співпадінь (рис. 3.1, б).

В обох випадках шкідливе програмне забезпечення було детектоване.

Уражений файл може містити мертвий код разом з тілом шкідливого програмного засобу, тому було прийнято рішення перевірити чи зможе алгоритм програмного засобу знайти уражений файл в якому міститься 100% тіла шкідливого програмного коду з мертвим кодом (рис. 3.1, в).

В обох випадках шкідливе програмне забезпечення було детектоване.

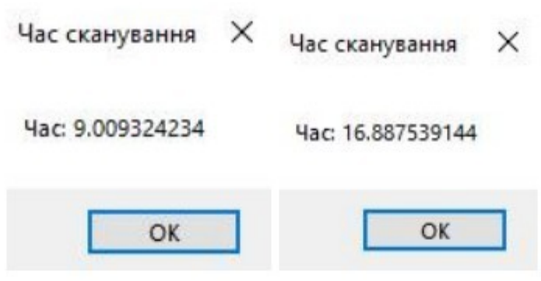
Також було перевірено чи зможе програмний засіб виявити підозрілий файл якщо він містить менш ніж 100% тестового вірусу Eicar, але більше допустимого рівня та містить мертвий код (рис. 3.1, г).

В обох випадках шкідливе програмне забезпечення було детектоване.

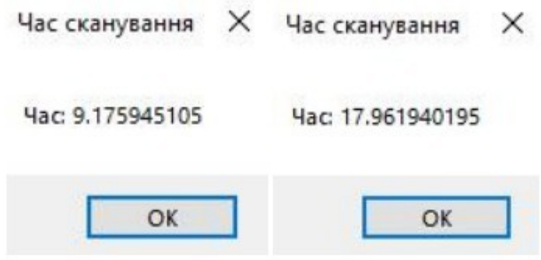
Також було прийнято рішення перевірити алгоритм програмного засобу чи зможе він виявити підозрілий файл який містить 100% з тестового вірусу Eicar, містить мертвий код та містить «опкод» команди jmp, для переходу з одної точки пам’яті файлу в іншу (рис. 3.1, є).

Якщо підозрілий файл не містить допустимого рівня шкідливого програмного коду, то файл не буде вважатись ураженим (рис. 3.1, ж).

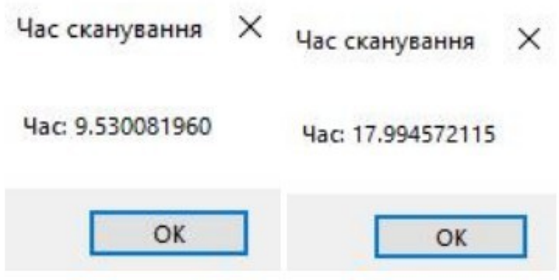
В обох випадках шкідливе програмне забезпечення було детектоване.



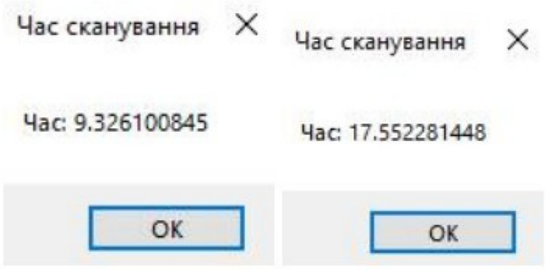
а)



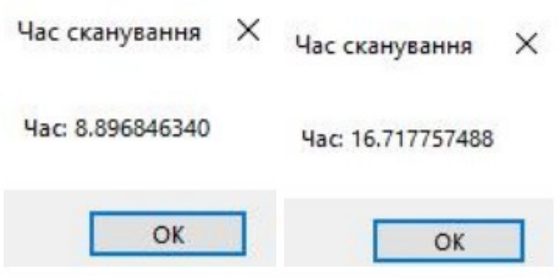
б)



в)



г)



є)

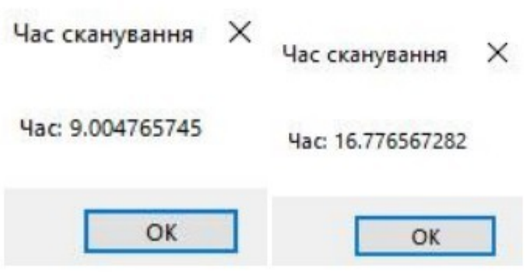


Рисунок 3.1 – Результати тестування (а – файл ураженим на 100% вірусом, б – файл ураженим вірусом на 85%, в – файл містив мертвий код, г – файл містив мертвий код та був ураженим на 85%, є – файл містив мертвий код та містив опкод команди jmp, ж – файл не уражений вірусом.

Тестування показало, що програмний засіб виконує покладені на нього функції, тобто детектує шкідливий програмний код. Для тестування було підібрано ряд тестових прикладів, які містять шкідливий програмний код на 100% і не містять його взагалі, містять мертвий код і опкоди команд переходу.

З результатів тестування можна зробити висновок, що метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз працює швидше ніж евристичний метод пошуку шкідливого програмного забезпечення на основі сигнатур.

Програмний засіб може бути використаний для швидкої перевірки комп'ютора користувача, не звертаючись за допомогою до громістких антивірусних програм.

3.4 Аналіз результатів тестування методів пошуку шкідливого програмного забезпечення

Було проведено тестування евристичного методу пошуку шкідливого програмного забезпечення на основі сигнатур та методу пошуку шкідливого програмного забезпечення шляхом прогнозування загроз, за результатами тестування стає зрозуміло, що метод шляхом прогнозування загроз є швидшим ніж евристичний метод на основі сигнатур. Також зрозуміло, що метод шляхом прогнозування загроз не доцільно використовувати у випадках коли потрібно проаналізувати невелику кількість даних, так як це займе більше часу ніж проводити аналіз евристичним методом на основі сигнатур, тому користувачу пропонується вибір, між двома методами пошуку шкідливого програмного забезпечення.

Для доцільності використання даного алгоритму було проаналізовані популярні алгоритми пошуку вірусів за критеріями: ресурсозатратність, фінансова складова, хибні спрацювання та виявлення шкідливого програмного забезпечення кожного з алгоритмів (табл. 1.3). Для оцінювання використано

власну шкалу, яка включає такі оцінки, як «високий», «середній» та «низький» рівень.

Таблиця 3.2 – Оцінювання алгоритмів пошуку вірусів

Метод/ алгоритм	Ресурсозатратність	Фінансова складова	Хибні спрацювання	Виявлення ШПЗ
Виявлення вірусів, що базується на сигнатурах	Високий	Середній	Низький	Середній
Виявлення аномалій	Середній	Середній	Високий	Середній
Виявлення вірусів, що базується на емуляції	Високий	Середній	Високий	Середній
Евристичний метод за базою сигнатур	Середній	Середній	Низький	Високий
Виявлення ШПЗ, шляхом прогнозування загроз	Низький	Середній	Низький	Високий

Після оцінювання кожного з методів видно, що всі методи ресурсозалежні, але не всі є такими надійними та дешевими, як евристичний метод за базою сигнатур.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

4.1 Аналіз комерційного потенціалу розробки (технологічний аудит розробки)методу пошуку ШПЗ шляхом прогнозування загроз.

4.1.1 Визначення рівня комерційного потенціалу розробки методу пошуку ШПЗ шляхом прогнозування загроз

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу методу пошуку ШПЗ шляхом прогнозування загроз. В результаті оцінювання можна буде зробити висновок щодо напрямів (особливостей) організації подальшого її впровадження з врахуванням встановленого рейтингу.

Для проведення технологічного аудиту залучимо 3-х незалежних експертів. У нашому випадку такими експертами будуть керівник магістерської роботи та провідні викладачі випускової та споріднених кафедр.

Оцінювання комерційного потенціалу розробки методу пошуку ШПЗ шляхом прогнозування загроз будемо здійснювати за 12-ю критеріями згідно рекомендацій.

Результати оцінювання комерційного потенціалу розробки методу пошуку ШПЗ шляхом прогнозування загроз заносимо до таблиці 4.1.

Таблиця 4.1. - Результати оцінювання комерційного успіху розробкиметоду пошуку ШПЗ шляхом прогнозування загроз

Критерії	Експерти		
	д. т. н., проф., Лужецький В.А.	к.т.н., ст. виклЛукічов В. В.	к.т.н., доцент Войтович О.П.
	Бали, виставлені експертами		
1	2	2	3
2	3	3	2

Продовження таблиці 4.1

3	4	4	3
4	4	2	3
5	4	3	3
6	4	4	3
7	3	3	2
8	3	3	3
9	3	4	4
10	2	3	3
11	3	3	3
12	2	3	3
Сума балів	37	37	35
Середньоарифметична сума балів, СБ	36		

За даними таблиці 4.1 робимо висновок щодо рівня комерційного потенціалу розробки методу пошуку ШПЗ шляхом прогнозування загроз. При цьому користуємося рекомендаціями, наведеними в таблиці 4.2.

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 50	Високий

Таким чином, робимо висновок, щодо рівня комерційного потенціалу нашої розробки методу пошуку ШПЗ шляхом прогнозування загроз вище середнього.

4.1.2 Визначення рівня якості розробки методу пошуку ШПЗ шляхом прогнозування загроз

Оцінювання рівня якості розробки методу пошуку ШПЗ шляхом прогнозування загроз проводиться з метою порівняльного аналізу і визначення найбільш ефективного, з технічної точки зору, варіанта інженерного рішення.

Рівень якості – це кількісна характеристика міри придатності певного виду продукції для задоволення конкретного попиту на неї при порівнянні з відповідними базовими показниками за фіксованих умов споживання.

Абсолютний рівень якості розробки адаптивного методу для автентифікації користувачів мобільних пристроїв знаходимо обчисленням вибраних для її вимірювання показників, не порівнюючи їх із відповідними показниками аналогічних виробів. Для цього необхідно визначити зміст основних функцій, які повинні реалізовувати розробка, вимоги замовника до неї, а також умови, які характеризують експлуатацію, визначають основні параметри, які будуть використані для розрахунку коефіцієнта технічного рівня виробу. Система параметрів, прийнята до розрахунків, повинна достатньо повно характеризувати споживчі властивості інноваційного товару (його призначення, надійність, економічне використання ресурсів, стандартизація тощо).

Далі визначаємо величину параметрів якості в балах та встановлюємо граничні його значення (кращі, гірші, середні). Всі ці дані для кожного параметра заносимо в табл. 4.3.

Таблиця 4.3 – Основні параметри методу пошуку ШПЗ шляхом прогнозування загроз

Параметри	Абсолютне значення параметра			Коефіцієнт вагомості параметра
	Краще +5…+4	Середнє +3	Гірше +1…+2	
Низька відноснапохибка			2	0,2

Безпека			2	0,5
Ресурсозатратність		3		0,2
Варіативність		3		0,1

Із врахуванням коефіцієнтів вагомості відповідних параметрів можна визначити абсолютний рівень якості інноваційного рішення за формулою:

$$K_{я.а.} = \sum_{i=1}^{n} Рні \bullet$$

ai, (4.1)

де Рні – числове значення і-го параметра інноваційного рішення, n – кількість параметрів інноваційного рішення, що прийняті для оцінювання, аі – коефіцієнт вагомості відповідного параметра (сума коефіцієнтів вагомості всіх параметрів повинна дорівнювати 1).

Отже, абсолютний рівень якості методу та засобу завадостійкого розподілу секрету становитиме – 2,3 бали.

Одночасно визначаємо відносний рівень якості методу пошуку ШПЗ шляхом прогнозування загроз, що виробляється (проектується), порівнюючи її показники з абсолютними показниками якості найліпших вітчизняних та зарубіжних аналогів (товарів-конкурентів) (табл. 4.4).

Таблиця 4.4 – Основні параметри методу пошуку ШПЗ шляхом прогнозування загроз та товару-конкурента

Параметри	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (конкурент)	Новий		
Низька відноснапохибка	2	1	2	0,2

Безпека	2	3	1,5	0,5

Продовження таблиці 4.4

Ресурсозатратність	5	4	0,8	0,2
Варіативність	3	2	0,7	0,1

Відносний рівень якості методу та засобу завадостійкого розподілу секрету визначаємо за формулою:

$$K_{я.в.} = \sum_{i=1}^{n} q i \bullet$$

ai, (4.2)

За розрахунками відносний рівень якості методу пошуку ШПЗ шляхом прогнозування загроз становитиме – 1,78. Це означає, що наша розробка краща за якістю на 78% від товару-аналога.

4.1.3 Визначення конкурентоспроможності розробки методу пошуку ШПЗ шляхом прогнозування загроз

У найширшому розумінні конкурентоспроможність товару – це можливість його успішного продажу на певному ринку і в певний проміжок часу. Для того, щоб високоякісний товар був одночасно і конкурентоспроможним, він має відповідати критеріям оцінювання споживачів конкретного ринку в конкретний час.

Дані для розрахунку загального показника конкурентоспроможності розробки необхідно занести до таблиці 4.5.

Таблиця 4.5 – Нормативні, технічні та економічні параметри методу пошуку ШПЗ шляхом прогнозування загроз і товару-конкурента

Параметри	Варіанти		Відносний показник	Коефіцієнт вагомості
	Базовий	Новий		

	(конкурент)		якості	параметра

Продовдження таблиці 4.5

Низька відносна похибка	2	1	2	0,2
Безпека	2	3	1,5	0,5
Ресурсозатратність	5	4	0,8	0,2
Варіативність	3	2	0,7	0,1
Ціна за продукт, тис. грн.	3000	2000	0,7	-

Загальний показник конкурентоспроможності розробки (К) з урахуванням вищезазначених груп показників визначаємо за формулою:

$$K = \frac{Iт.п.}{Ie.п.} = \frac{1,78}{0,7} = 2,5, \quad (4.3)$$

де Іт.п. – індекс технічних параметрів (відносний рівень якості інноваційного рішення); Іе.п. – індекс економічних параметрів.

$$Ie.\,п. = \frac{PHei}{PБei} = \frac{2000}{3000} = 0,7, \quad (4.4)$$

де РНеі, РБеі – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Згідно розрахунків загальний показник конкурентоспроможності –2,5. Це означає, що наша розробка методу пошуку ШПЗ шляхом прогнозування загроз більш конкурентна майже в 2,5 рази від товару-аналога.

4.2 Прогнозування витрат на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи

4.2.1 Розрахунок витрат, що стосуються виконавців розробки методу пошуку ШПЗ шляхом прогнозування загроз

Основна заробітна плата кожного із розробників (дослідників) Зо, якщо вони працюють в наукових установах бюджетної сфери:

$$Зо = \frac{М}{Тр} \bullet$$

$$t, \qquad (4.5)$$

де М – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.

У 2019 році величини окладів (разом з встановленими доплатами і надбавками) рекомендується брати в межах (5000…10000) грн. за місяць; Тр – число робочих днів в місяці; приблизно Тр = (21…23) дні; t – число робочих днів роботи розробника (дослідника).

Зроблені розрахунки зводимо до таблиці 4.5.

Таблиця 4.6 – Заробітна плата розробників

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	8000	381	5	1905
Інженер-програміст	4500	214	5	1070
Всього:				2975

Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою:

$$Зр = \sum_{i=1}^{n} t i \bullet Ci,$$

(4.6)

де ti – норма часу (трудомісткість) на виконання конкретної роботи, годин; n – число робіт по видах та розрядах; Сі – погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу. Сі визначається за формулою:

$$Ci = \frac{Мм \bullet Ki}{Тр \bullet Тзм}, \quad (4.7)$$

де Мм – розмір мінімальної заробітної плати за місяць, грн.; в 2019 році мінімальна заробітна плата становить – 4173 грн., Кі – тарифний коефіцієнт робітника відповідного розряду, Тр – число робочих днів в місяці; приблизно Тр = 21…23 дні; Тзм – тривалість зміни, зазвичай Тзм = 8 годин.

Таблиця 4.7 – Заробітна плата робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Програмувальні	7	5	34	1,36	170
Всього					170

Додаткова заробітна плата Зд всіх розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = 0,1 \bullet (Зр + Зо) = 0,1 \bullet (2975 + 170) = 314,5 \ грн.$$

(4.8)

Нарахування на заробітну плату Нзп розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

де Зо – основна заробітна плата розробників, грн.; Зр – основна заробітна плата робітників, грн.; Зд – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов’язкове державне

соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

$$Нзп = 0,22 \bullet (Зр + Зо + Зд) = 0,22 \bullet (2975 + 170 + 314,5) = 761\ грн.$$

Амортизація обладнання, комп'ютерів та приміщень А, які використовувались під час (чи для) виконання даного етапу роботи.

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

У спрощеному вигляді амортизаційні відрахування А в цілому бути розраховані за формулою:

$$A = \frac{Ц \bullet На}{100} \bullet \frac{Т}{12}, \qquad (4.10)$$

де Ц – загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн.; На – річна норма амортизаційних відрахувань. Для нашого випадку можна прийняти, що На = (10...25)%; Т – термін, використання обладнання, приміщень тощо, місяці.

Таблиця 4.8 - Амортизаційні відрахування

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
ПК	7000	20	2	233
Іншеобладнання	3000	10	1	25
Всього	258			

Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = В \bullet П \bullet Ф \bullet Кп,\ грн \qquad (4.11)$$

В – вартість 1 кВт-год. електроенергії, в 2019 р. В ≈ 8,45 грн./кВт; П – установлена потужність обладнання, кВт; Ф – фактична кількість годин роботи обладнання, годин, Кп – коефіцієнт використання потужності; Кп< 1.

Потужність обладнання складає –0,5 кВт.

Кількість годин роботи складає – 100годин.

Коефіцієнт викор. потужності -0,9.

Ве=380 грн.

Інші витрати Він охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати Ів можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$Ів = 1{,}5 \bullet (Зо + Зр) = 1{,}5 \bullet (2975 + 170) = 4718\ грн. \qquad (4.12)$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини (розділу, етапу) роботи – В.

$$В = 9577 \text{ грн.}$$

4.2.2 Розрахунок собівартості розробки методу пошуку ШПЗ шляхом прогнозування загроз

Витрати на силову електроенергію Ве, якщо ця стаття має суттєве значення для виконання даного етапу роботи, розраховуються за формулою:

$$Ве = В \bullet П \bullet Ф \bullet Кп,\ грн$$

В – вартість 1 кВт-год. електроенергії, в 2019 р. В ≈ 8,45 грн./кВт; П – установлена потужність обладнання, кВт; Ф – фактична кількість годин роботи обладнання, годин, Кп – коефіцієнт використання потужності; Кп< 1.

Потужність обладнання складає – 0,5 кВт.

Кількість годин роботи складає – 100 годин.

Коефіцієнт викор. потужності -0,9.

Ве=380 грн.

Основна заробітна плата робітників Зр, якщо вони беруть участь у виконанні даного етапу роботи і виконують роботи за робочими професіями у випадку, коли вони працюють в наукових установах бюджетної сфери, розраховується за формулою:

$$Зр = \sum_{i=1}^{n} t i \bullet$$

$Ci,$ (4.12)

де ti – норма часу (трудомісткість) на виконання конкретної роботи, годин; n – число робіт по видах та розрядах; Сі – погодинна тарифна ставка робітника відповідного розряду, який виконує дану роботу. Сі визначається за формулою:

$$Ci = \frac{Мм \bullet Ki}{Тр \bullet Тзм}, \quad (4.13)$$

де Мм – розмір мінімальної заробітної плати за місяць, грн.; в 2019 році мінімальна заробітна плата становить – 4173 грн., Кі – тарифний коефіцієнт робітника відповідного розряду, Тр – число робочих днів в місяці; приблизно Тр = 21…23 дні; Тзм – тривалість зміни, зазвичай Тзм = 8 годин.

Таблиця 4.9 – Заробітна плата робітників

Найменування робіт	Трудомісткість, н-год.	Розряд роботи	Погодинна тарифна ставка	Тариф. коеф.	Величина, грн.
Програмувальні	7	5	34	1,36	170
Всього					170

Додаткова заробітна плата Зд всіх робітників, які брали участь у виконанні даного етапу роботи, розраховується як (10…12)% від суми основної заробітної плати всіх розробників та робітників, тобто:

$$Зд = 0,1 \bullet (Зо) = 0,1 \bullet (170) = 17 \ грн.$$

(4.14)

Нарахування на заробітну плату Нзп розробників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою:

де Зо – основна заробітна плата розробників, грн.; Зр – основна заробітна плата робітників, грн.; Зд – додаткова заробітна плата всіх розробників та робітників, грн.; β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % (приймаємо для 1-го класу професійності ризику 22%).

$$Нзп = 0,22 \bullet (Зо + Зд) = 0,22 \bullet (170 + 17) = 41 \ грн.$$

(4.15)

«Загальновиробничі витрати» належать витрати: пов'язані з управлінням виробництвом (утримання працівників апарату управління виробництвом, оплата службових відряджень персоналу цехів, витрати на інформаційне забезпечення управління тощо); на повне відновлення та капітальний ремонт основних фондів загальновиробничого призначення; витрати некапітального характеру, пов'язані з удосконаленням технологій та організацією виробництва, поліпшенням якості продукції; на утримання, обслуговування, поточний ремонт виробничих приміщень; на контроль за виробничими процесами та кістю продукції.

Крім того, загальновиробничі витрати з розрахунку на одиницю продукції можна розрахувати за нормативами відносно до основної заробітної плати основних робітників, які виготовляють продукцію:

$$ЗВВ = Нв \bullet Зо,$$

(4.16)

Норматив загальновиробничих витрат для програмних продуктів становить 230-270%.

$$ЗВВ = 2,5 \bullet 170 = 425 \ грн,$$

Сума попередніх витрат утворює виробничу собівартість розробки:

$$S_в = 1033 \ грн.$$

4.3 Розрахунок ціни та чистого прибутку від реалізації розробки методу пошуку ШПЗ шляхом прогнозування загроз

Ціна – це грошовий вираз вартості товару (продукції, послуги). Вона завжди коливається навколо ціни виробництва (перетвореної форми вартості одиниці товару, що дорівнює сумі витрат виробництва й середнього прибутку) та відображає рівень суспільне необхідних витрат праці.

Виходячи з того, що розробки, як правило, приймаються та впроваджуються за завданням замовника, або коли результатом розробки є продукція, що підлягає державному регулюванню, то нижню межу ціни реалізації розробки можна розрахувати за формулою:

$$Ц = S_в \bullet \left(1 + \frac{P}{100}\right) \bullet (1 + \frac{\omega}{100}), \quad (4.17)$$

де Sв – виробнича собівартість інноваційного рішення, грн.; Р – норматив рентабельності узгоджений із замовником або встановлений державою, (Р=30…60%); w – ставка податку на додану вартість, % (в 2019 році w=20%).

$$Ц = 1033 \bullet \left(1 + \frac{60}{100}\right) \bullet \left(1 + \frac{20}{100}\right) = 1983 \ грн.$$

Чистий прибуток від реалізації розробки можна розрахувати за формулою:

$$П = \left(Ц - \frac{(Ц - МР) \bullet f}{100} - S_в - \frac{q \bullet S_в}{100}\right) \bullet (1 - \frac{h}{100}) \bullet РП, \quad (4.18)$$

де Ц – ціна розробки, грн.; МР – вартість матеріальних та інших ресурсів, що були придбані виробником для виготовлення розробки (МР=(0,1…0,2) Цр), грн.; f – зустрічна ставка податку на додану вартість, %; Sв – виробнича собівартість розробки, грн.; q – норматив, який визначає величину

адміністративних витрат, витрат на збут та інші операційні витрати, % (рекомендовано q=5…10%); h – ставка податку на прибуток, %, РП – прогнозований попит продажів:

$$П = 8360 \text{ грн.}$$

4.4 Розрахунок терміну окупності коштів, вкладених в наукову розробку методу пошуку ШПЗ шляхом прогнозування загроз

Термін окупності вкладених у реалізацію наукового проекту інвестицій Ток можна розрахувати за формулою:

$$Ток = \frac{В}{П} = \frac{9577}{8360} = 1,15 \ роки.$$

(4.19)

Оскільки Ток < 3 років, то фінансування даної наукової розробки методу пошуку ШПЗ шляхом прогнозування загроз.

ВИСНОВКИ

Магістерська кваліфікаційна робота присвячена проблемі захисту комп'ютерів від шкідливого програмного забезпечення.

Аналіз інформаційних джерел показав, що існує певна кількість методів пошуку шкідливого програмного. Кожний з методів має свої позитивні та негативні риси. Наведено класифікацію методів пошуку шкідливого програмного забезпечення, розглянуто кожний з можливих способів захисту. Найкращі показники захищеності показують комбіновані методи пошуку.

Тому в результаті аналізу сучасного стану проблеми було вирішено використовувати комбінований метод пошуку, а саме евристичний метод на основі сигнатур.

Другий розділ пояснювальної записки присвячений обґрунтування вибору евристичного методу на основі сигнатур та комбінування його з нейромережею, вибору мови програмування та розробці програмного засобу для пошуку шкідливого програмного забезпечення.

Результатом виконання магістерської кваліфікаційної роботи є програмний засіб, виконаний у середовищі Visual Studio мовою C#. Для розробки засобу підготовлений ряд схем і алгоритмів, розроблений інтерфейс, реалізовано цілу низку окремих функцій для виконання конкретних операцій.

Розроблений програмний засіб може бути використаний для захисту комп'ютерів від шкідливого програмного забезпечення різних форматів, txt, exe, bat, тощо.

Недоліком магістерської кваліфікаційної роботи є те, що сама програма не є захищеною від дослідження, а тому алгоритми захисту можуть бути досліджені. Це стане проблемою подальшого опрацювання. Іншим недоліком можна вважати те, що не проведено перевірку на стійкість до зламу.

Разом з тим розробка може бути покращена шляхом створення захищеної бази сигнатур. В цілому, завдання на бакалаврську роботу виконано в повному обсязі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Laurence. Viruses that can cost you. www.symantec.com. Процитовано 2017-05-23.
2. Кримінальний кодекс України : закон України від 23 грудня 2004 р. № 2289-IУ// Відомості Верховної Ради України. – 2004. – № 30. – Ст. 361.
3. Шкідливе програмне забезпечення [Електронний ресурс]. – Режим доступу : URL : https://moikomputer.ru/ – Назва з екрану.
4. Шкідливе програмне забезпечення [Електронний ресурс]. – Режим доступу : URL : https://www.hipaajournal.com – Назва з екрану.
5. Аналіз антивірусів [Електронний ресурс]. – Режим доступу : URL : https://sites.google.com – Назва з екрану.
6. Статистичні дані за 2017 рік [Електронний ресурс]. – Режим доступу : URL : https://securelist.ru/- Назва з екрану.
7. Moore, Gordon E. (1965). "Cramming more components onto integrated circuits" (PDF). Electronics Magazine. p. 4. Retrieved 2006-11-11
8. Denning, D. E. (1987). An Intrusion-Detection Model. IEEE Transactions on Software Engineering (2): 222. doi: 10.1109/TSE.1987.232894. CiteSeerX: 10.1.1.102.5127.
9. Лукічов В. В. Методичні вказівки до самостійної роботи студентів при вивчені дисципліни «Технології програмування» зі спеціальності 125 «Кібербезпека» / В. В. Лукічов, А. С. Васюра – Вінниця, 2010. – 8 с.
10. Виявлення, засноване на емуляції / Нац. ун-т "Львів. політехніка" ; [відп. ред. К. Р. Третяк]. – Львів : Вид-во Нац. ун-ту "Львів. політехніка", 2008. – Вип. 70. – 88 с.
11. Geier, Eric (2012-01-16). "How to Keep Your PC Safe With Sandboxing". TechHive. Retrieved 2014-07-03.
12. Петров А. П. О возможностях перцептрона // Известия АН СССР, Техническая кибернетика. — 1964. — № 6.

13. Romanuke, Vadim. Appropriate number and allocation of ReLUs in convolutional neural networks (англ.) // Research Bulletin of NTUU “Kyiv Polytechnic Institute” : journal. — 2017. — Vol. 1. — P. 69—78.
14. B.V.Kryzhanovsky, B.M.Magomedov, A.L.Mikaelian. «A Domain model of neural network», Doklady Mathematics vol.71, pp. 310–314 (2005).
15. Yin H. Learning nonlinear principal manifolds by self-organising maps, In: Gorban A. N. et al (Eds.), LNCSE 58, Springer, 2007.
16. BillWagner. virtual (C# Reference). docs.microsoft.com (en-us). Процитовано 2018-05-07.
17. Stallman, Richard (2009-06-26). Why free software shouldn't depend on Mono or C#. Free Software Foundation. Архів оригіналу за 2013-06-22. Процитовано 2009-07-02.
18. The ECMA C# and CLI Standards. 2009-07-06. Архів оригіналу за 2013-06-22. Процитовано 2009-078-03.
19. Вибір тестів для перевірки [Електронний ресурс]. – Режим доступу : URL : www.comss.ru/page.php?id=4243- Назва з екрану.
20. База сигнатур ClamAV [Електронний ресурс]. – Режим доступу : URL : https://www.clamav.net/- Назва з екрану.
21. Тестовий вірус Eicar [Електронний ресурс]. – Режим доступу : URL : http://www.eicar.org/- Назва з екрану.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра захисту інформації

ЗАТВЕРДЖУЮ

Завідувач кафедри ЗІ, д. т. н., проф.

_____ В. А. Лужецький

"____" _____ 2019 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до магістерської кваліфікаційної роботи на тему:

"Метод пошуку шкідливого програмного забезпечення"

08-20.МКР.011.00.000 ТЗ

Розробив студент групи 1БС-18м

_____ Новотарський О. Ю.

Керівник магістерської кваліфікаційної роботи к. т. н., ст. викл

_____ Лукічов В. В.

_____ _____ 2019 р.

Вінниця 2019

1 Найменування та область застосування

Метод пошуку шкідливого програмного забезпечення. Область застосування: розробка антивірусних засобів.

2 Підстави для розробки

Розробка виконується на основі наказу № 254 ректора ВНТУ від 02.10.2019 р..

3 Мета та призначення

Підвищення ефективності захисту комп’ютерних систем і мереж від несанкціонованого ураження файлів вірусами.

4 Джерела розробки

4.1 Laurence. Viruses that can cost you. www.symantec.com. Процитовано 2017-05-23.

4.2 Кримінальний кодекс України : закон України від 23 грудня 2004 р. № 2289- IУ// Відомості Верховної Ради України. – 2004. – № 30. – Ст. 361.

4.3 Шкідливе програмне забезпечення [Електронний ресурс]. – Режим доступу : URL : https://moikomputer.ru/ – Назва з екрану.

4.4 Шкідливе програмне забезпечення [Електронний ресурс]. – Режим доступу : URL : https://www.hipaajournal.com – Назва з екрану. 2015. – № 24. – 171-176 с.

5 Технічні вимоги

5.1 Параметри розроблюваної системи захисту:

- механізм захисту – автономний додаток;
- принцип захисту – захист від несанкціонованого ураження файлів;
- метод захисту – евристичний аналізатор;
- об’єкт захисту – файли на комп'ютері.

5.2 Програма повинна працювати без помилок і давати змогу:

- програма повинна шукати віруси;
- отримувати доступ до інформації при виконанні відповідних вимог;
- користуватись динамічною системою підказок.

5.4 Вимоги до апаратного і програмного забезпечення, на якому повинна працювати програма:

- процесор – Pentium, Athlon і сумісні з ними;
- оперативна пам’ять – не менше 512 Мбайт;
- середовище функціонування – ОС сімейства Windows, починаючи з Windows XP.

5.5 Вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп'ютерною технікою.

6 Стадії та етапи розробки

№	Зміст	Початок	Закінчення	Результат
1	Вступ. Розробка ТЗ. Огляд літературних джерел	01.09.2019	22.09.2019	Розділ пояснювальної записки, технічне завдання
2	Розробка системи, моделі, алгоритму, структури	23.09.2019	12.10.2019	Розділ пояснювальної записки
3	Програмна реалізація	14.10.2019	17.11.2019	Модулі програмного засобу
	Тестування засобу	18.11.2019	24.11.2019	Діюча програма. Пояснювальна записка
4	Аналіз виконання ТЗ, висновки. Оформлення пояснювальної записки	25.11.2019	30.11.2019	Пояснювальна записка

7 Порядок контролю та приймання

7.1 До приймання дипломної роботи представляється:

– остаточний звіт (пояснювальна записка);

– робоча програма;

– результати функціонального тестування програми на ефективність захисту;

– ілюстративні матеріали.

7.2 Рубіжний контроль керівника _____

7.3 Попередній захист на кафедрі _____

7.4 Захист на ДЕК _____

Додаток Б

Лістинг програмного засобу

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace antiVirus
{
    static class Program
    {
        public static Form1 Form;
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Form = new Form1();
            Application.Run(Form);
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using System.Text.RegularExpressions;
using System.Collections;
using System.Media;
using System.Xml;
namespace antiVirus
{
    public partial class Form1 : Form
    {
Function function = new Function();
//Options op = new Options();
int count;
float KilkictbVsixPorivnyanb = 0;
float KilkictbSpivpadinb = 0;
float VidsotkoveSpivvidnishenyaSpivpaninya = 0;
string FileNotFound = "";
string Error = "";
string Firstselectthedirectory = "";
string TypeOfFiles = "*.*";
int rate = 0;
public Form1()
{
    InitializeComponent();
```

```
}
private void button2_Click(object sender, EventArgs e)
{
    XmlElement xRoot = function.downloadXml();
    folderBrowserDialog1.ShowDialog();
    lblfolder.Text = folderBrowserDialog1.SelectedPath;
    count = 0;
    lblviruses.Text = function.getAtribut("Viruses", xRoot) + count.ToString();
    progressBar1.Value = 0;
    listBox1.Items.Clear();
}

private void button1_Click(object sender, EventArgs e)
{
    XmlElement xRoot = function.downloadXml();
    listBox1.Items.Clear();
    FileNotFound = function.getAtribut("FilenotFound", xRoot);
    Error = function.getAtribut("Error", xRoot);
    Firstselectthedirectory = function.getAtribut("Firstselectthedirectoryandtryagain", xRoot);
    //string ViruSList;//
    string Debug = Directory.GetCurrentDirectory();
    List<string> searchSignatur = Directory.GetFiles(Debug, "*.txt*").ToList();
    List<string> AllFoundViruses = new List<string>();
    List<string> searchDerictories = null;

    try
    {
        searchDerictories = Directory.GetDirectories(@folderBrowserDialog1.SelectedPath, "*", SearchOption.AllDirectories).ToList();
    }
    catch (ArgumentException ax)
    {
        MessageBoxButtons buttons = MessageBoxButtons.OK;
        MessageBox.Show(Firstselectthedirectory, Error, buttons);
        return;
    }
    searchDerictories.Add(@folderBrowserDialog1.SelectedPath);
    foreach (string signatur in searchSignatur)
    {
        try
        {
            int endIndexforListBox = folderBrowserDialog1.SelectedPath.ToString().Length;
            foreach (string directories in searchDerictories)
            {
                List<string> search = Directory.GetFiles(directories, TypeOfFiles).ToList();
                progressBar1.Maximum = search.Count;
                int flagok = 0;
                foreach (string item in search)
                {
                    foreach (string virus in AllFoundViruses)
                    {
                        if (virus == item.Substring(endIndexforListBox + 1, item.Length - endIndexforListBox - 1)) flagok++;
                    }
                    if (flagok > 0)
                    {
                        flagok = 0;
                        continue;
                    }

                    function.theFileIsScaned(item);
                    KilkictbVsixPorivnyanb = 0;
                    KilkictbSpivpadinb = 0;
                    try
                    {

                        string ViruSList = System.IO.File.ReadAllText(@signatur, Encoding.Default).Replace("\n", " ").Replace("\r", "");
```

```
                    string[] VirusListString = ViruSList.Split();
                    string Fileread = System.IO.File.ReadAllText(@item, Encoding.Default).Replace("\n", "").Replace("\r", " ");//
ReadFile(signatur, xRoot);
                    string[] read = Fileread.Split();
                    function.algoritmString(VirusListString, read, ref KilkictbVsixPorivnyanb, ref KilkictbSpivpadinb);
                    VidsotkoveSpivvidnishenyaSpivpaninya = KilkictbSpivpadinb / KilkictbVsixPorivnyanb;
                    function.virusCheck(VidsotkoveSpivvidnishenyaSpivpaninya, ref count, xRoot, item, AllFoundViruses,
endIndexforListBox);
                    if (VidsotkoveSpivvidnishenyaSpivpaninya > 0.85)
                    {
                        VidsotkoveSpivvidnishenyaSpivpaninya = 0;
                        KilkictbSpivpadinb = 0;
                        progressBar1.Increment(1);
                        continue;
                    }
                    VidsotkoveSpivvidnishenyaSpivpaninya = 0;
                    KilkictbSpivpadinb = 0;
                    progressBar1.Increment(1);

                    Fileread = Fileread.Replace(" ", "");
                    ViruSList = ViruSList.Replace(" ", "");
                    byte[] Virus = System.Text.Encoding.ASCII.GetBytes(Fileread);
                    byte[] Signatur = System.Text.Encoding.ASCII.GetBytes(ViruSList);
                    function.algoritmWithJmp(Signatur, Virus, ref KilkictbVsixPorivnyanb, ref KilkictbSpivpadinb,
VidsotkoveSpivvidnishenyaSpivpaninya, ref count, xRoot, item, AllFoundViruses, endIndexforListBox);
                    VidsotkoveSpivvidnishenyaSpivpaninya = 0;
                    progressBar1.Increment(1);
                }
                catch (Exception ex)
                {
                    MessageBoxButtons buttons = MessageBoxButtons.OK;
                    MessageBox.Show(FileNotFound, Error, buttons);
                    return;
                }

            }
        }
        catch (ArgumentException ax)
        {
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            MessageBox.Show(Firstselectthedirectory, Error,buttons);
            return;
        }
    }
}
private void exitToolStripMenuItem1_Click(object sender, EventArgs e)
{
    this.Close();
}

private void aboutToolStripMenuItem1_Click(object sender, EventArgs e)
{
    About form2About = new About();
    form2About.Show();
}

private void instructionToolStripMenuItem1_Click(object sender, EventArgs e)
{
    Instruction form3Instruction = new Instruction();
    form3Instruction.Show();
}

private void optionsToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```
XmlElement xRoot = function.downloadXml();
Options form4Options = new Options();
form4Options.groupBox1.Text = function.getAtribut("Newsignature", xRoot);
form4Options.label1.Text = function.getAtribut("Filename", xRoot);
form4Options.button3.Text = function.getAtribut("Add", xRoot);
form4Options.checkBox1.Text = function.getAtribut("Sound", xRoot);
form4Options.groupBox2.Text = function.getAtribut("Language", xRoot);
form4Options.button2.Text = function.getAtribut("Ok", xRoot);
form4Options.button1.Text = function.getAtribut("Closed", xRoot);
form4Options.Text = function.getAtribut("Options", xRoot);
form4Options.groupBox2.Text = function.getAtribut("Files", xRoot);
form4Options.radioButton4.Text = function.getAtribut("AllFiles", xRoot);
form4Options.Show();
   if (TypeOfFiles == "*.exe") form4Options.radioButton1.Checked = true;
   if (TypeOfFiles == "*.txt") form4Options.radioButton2.Checked = true;
   if (TypeOfFiles == "*.bat") form4Options.radioButton3.Checked = true;
   if (TypeOfFiles == "*.*") form4Options.radioButton4.Checked = true;
   if (form4Options.GetMusic() == "Checked") form4Options.checkBox1.Checked = true;
   rate = form4Options.rate;
}

private void infoToolStripMenuItem_Click(object sender, EventArgs e)
{

}

private void folderBrowserDialog1_HelpRequest(object sender, EventArgs e)
{

}
public string GetTypeOfFile()
{
      return this.TypeOfFiles;
   }
   public void SetTypeOfFile(string typeoffiles)
   {
      this.TypeOfFiles = typeoffiles;
   }
  }
}
```

Options.cs

```
using System;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;
using System.Xml;

namespace antiVirus
{
   public partial class Options : Form
{
   Function func = new Function();
   public static string Music = "";
   public static string nameXML = "English";
   public int rate;
   //public string rate
   public Options()
   {
      InitializeComponent();
   }

   private void button1_Click(object sender, EventArgs e)
   {
      this.Close();
```

```
}

private void button3_Click(object sender, EventArgs e)
{
    string textbox = textBox1.Text;

    XmlElement xRoot = func.downloadXml();
    string Warning;
    string EnterName;
    string fileWithThisNameAlreadyExists;
    Warning = func.getAtribut("Warning", xRoot);
    EnterName = func.getAtribut("EnterName", xRoot);
    fileWithThisNameAlreadyExists = func.getAtribut("fileWithThisNameAlreadyExists", xRoot);
    if (!System.IO.File.Exists(textbox + ".txt"))
    {
        if (textbox == "")
        {
            MessageBoxButtons buttons = MessageBoxButtons.OK;
            MessageBox.Show(EnterName, Warning, buttons);
            return;
        }
        System.IO.FileStream fs = System.IO.File.Create(textbox + ".txt");
        fs.Close();
        Process.Start(textbox + ".txt");
    }
    else
    {
        MessageBoxButtons buttons = MessageBoxButtons.OK;
        MessageBox.Show(fileWithThisNameAlreadyExists, Warning, buttons);
        return;
    }
}


private void button2_Click(object sender, EventArgs e)
{

    string unCorrect;
    string error;
    nameXML = comboBox1.SelectedItem.ToString();
    XmlElement xRoot = func.downloadXml();
    Music = checkBox1.CheckState.ToString();
    Program.Form.Text = func.getAtribut("AntiVirus", xRoot);
    Program.Form.infoToolStripMenuItem.Text = func.getAtribut("Information", xRoot);
    Program.Form.optionsToolStripMenuItem.Text = func.getAtribut("Options", xRoot);
    Program.Form.label1.Text = func.getAtribut("Infectedfiles", xRoot);
    Program.Form.lblfolder.Text = func.getAtribut("Folder", xRoot);
    Program.Form.lblfolder.Text = func.getAtribut("Folder", xRoot);
    Program.Form.button1.Text = func.getAtribut("Scan", xRoot);
    Program.Form.button2.Text = func.getAtribut("Browse", xRoot);
    Program.Form.lblviruses.Text = func.getAtribut("Viruses", xRoot);
    Program.Form.labelFile.Text = func.getAtribut("Thefileisscanned", xRoot);
    Program.Form.aboutToolStripMenuItem1.Text = func.getAtribut("About", xRoot);
    Program.Form.instructionToolStripMenuItem1.Text = func.getAtribut("Instruction", xRoot);
    Program.Form.exitToolStripMenuItem1.Text = func.getAtribut("Exit", xRoot);
    if (radioButton1.Checked) Program.Form.SetTypeOfFile("*.exe");
    if (radioButton2.Checked) Program.Form.SetTypeOfFile("*.txt");
    if (radioButton3.Checked) Program.Form.SetTypeOfFile("*.bat");
    if (radioButton4.Checked) Program.Form.SetTypeOfFile("*.*");

    error = func.getAtribut("Error", xRoot);
    unCorrect = func.getAtribut("unCorrect", xRoot);
    if (!Int32.TryParse(textBox2.Text, out rate))
    {
```

```
                MessageBox.Show(unCorrect, error, MessageBoxButtons.OK);
                return;
            }else if (rate > 100 || rate < 0)
            {
                MessageBox.Show(unCorrect, error, MessageBoxButtons.OK);
                return;
            }
            this.Close();
        }

    }
}
```

Function.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace antiVirus
{
    class Function
    {

        public void algoritmString(string [] VirusListString, string[] read, ref float KilkictbVsixPorivnyanb, ref float KilkictbSpivpadinb)
        {
            //Program.
int i = 0;
KilkictbVsixPorivnyanb = VirusListString.Length;
for (; i < VirusListString.Length; i++)
{
    for (int j = 0; j < read.Length; j++)
    {

        if (i == VirusListString.Length) break;
        if (VirusListString[i] == read[j])
        {
            KilkictbSpivpadinb++;
            int k = i + 1;
            int z = j + 1;
            int memory = i;
            for (; k < VirusListString.Length - memory; k++)
            {
                if (VirusListString[k] == read[z])
                {
                    KilkictbSpivpadinb++;
                    i = k + 1;
                }
                if (z + 1 == read.Length)
                {
                    break;
                }
                z++;
            }
            j = -1;
            if (memory == i) i += 1;

        }
    }
}
```

```
}
public string ReadFile(string NameFile, XmlElement xRoot)
{
    string FileNotFound = "";
    string Error = "";
    string s = "";
    FileNotFound = getAtribut("FilenotFound", xRoot);
    Error = getAtribut("Error", xRoot);
    try
    {
        using (StreamReader stream = new StreamReader(NameFile, System.Text.Encoding.Default))
        {
            while (!stream.EndOfStream)
                s += stream.ReadLine();
        }
    }
    catch (Exception ex)
    {
        MessageBoxButtons buttons = MessageBoxButtons.OK;
        MessageBox.Show(FileNotFound, Error, buttons);
        return null;
    }
    return s;
}
public void VoicePig()
{
    if (Options.Music == "Checked")
    {
        //Options.Music
        string path = "pig.wav";
        System.Media.SoundPlayer player = new System.Media.SoundPlayer();
        player.SoundLocation = path;
        player.Load();
        player.Play();
    }
}
public XmlElement downloadXml()
{
    XmlDocument xDoc = new XmlDocument();
    xDoc.Load(Options.nameXML + ".xml");
    XmlElement xRoot;
    return xRoot = xDoc.DocumentElement;
}
public bool virusCheck(float VidsotkoveSpivvidnishenyaSpivpaninya, ref int count, XmlElement xRoot, string item, List<string>
AllFoundViruses, int endIndexforListBox)
{
    bool flag = true;
    if (VidsotkoveSpivvidnishenyaSpivpaninya > 0.85)
    {
        VoicePig();
        count++;
        Program.Form.lblviruses.Text = getAtribut("Viruses", xRoot) + count.ToString();
        Program.Form.lblviruses.Refresh();
        int endIndexListBox = item.Length;
        Program.Form.listBox1.Items.Add(item.Substring(endIndexforListBox + 1, endIndexListBox - endIndexforListBox - 1));
        Program.Form.listBox1.Refresh();
        AllFoundViruses.Add(item.Substring(endIndexforListBox + 1, endIndexListBox - endIndexforListBox - 1));
        flag = true;
    }
    else flag = false;
    return flag;
}
public void theFileIsScaned(string item)
{
    int startIndex = item.LastIndexOf('\\');
```

```
    int endIndex = item.Length;
    Program.Form.label2.Text = "..." + item.Substring(startIndex + 1, endIndex - startIndex - 1);
    Program.Form.label2.Refresh();
}

public string getAtribut(string s, XmlElement Root)
{
    foreach (XmlNode xnode in Root)
    {
        if (xnode.Attributes.Count > 0)
        {
            XmlNode attr = xnode.Attributes.GetNamedItem(s);
            if (attr != null)
                return attr.Value;
        }
    }
    return null;
}


public void algoritmWithJmp2(byte[] VirusListString, byte[] read, int i, int j, ref float KilkictbSpivpadinb)
{
    for (; j <read.Length;)
    {
        if (read[j] == 233 || read[j] == 234 || read[j] == 235)
        {
            j = 0;
            j = algoritmWithJmp3(VirusListString, read, i, j);
        }
        if (VirusListString[i] == read[j])
        {
            KilkictbSpivpadinb++;
            i++;
            j++;
        }
        else{

            j = algoritmWithJmp3(VirusListString, read, i, j);

        }
        if (j == read.Length || i == VirusListString.Length) return;

    }
}

public int algoritmWithJmp3(byte[] VirusListString, byte[] read, int i, int j)
{
    for (; j <read.Length;)
    {
        if (VirusListString[i] == read[j])
        {
            return j;
        }
        if (j + 1 == read.Length || i + 1 == VirusListString.Length) return j;
        j++;
    }
    return j;
}


    public void algoritmWithJmp(byte[] VirusListString, byte[] read, ref float KilkictbVsixPorivnyanb, ref float KilkictbSpivpadinb, float
VidsotkoveSpivvidnishenyaSpivpaninya, ref int count, XmlElement xRoot, string item, List<string> AllFoundViruses, int endIndexforListBox)
    {
        if (VirusListString.Length > read.Length) return;
```

```
int f = 0;
KilkictbVsixPorivnyanb = VirusListString.Length;
for (int i = 0; i < VirusListString.Length; i++)
{
    for (int j = 0; j < read.Length; j++)
    {

        if (read[j] == VirusListString[i])
        {
            algoritmWithJmp2(VirusListString, read, i, j, ref KilkictbSpivpadinb);
            VidsotkoveSpivvidnishenyaSpivpaninya = KilkictbSpivpadinb / KilkictbVsixPorivnyanb;
            if (virusCheck(VidsotkoveSpivvidnishenyaSpivpaninya, ref count, xRoot, item, AllFoundViruses, endIndexforListBox)) return;
            KilkictbSpivpadinb = 0;
        }
        else if (read[j] == 233 || read[j] == 234 || read[j] == 235)
        {
            f = 0;
            f = algoritmWithJmp3(VirusListString, read, i, f);
            algoritmWithJmp2(VirusListString, read, i, f, ref KilkictbSpivpadinb);
            VidsotkoveSpivvidnishenyaSpivpaninya = KilkictbSpivpadinb / KilkictbVsixPorivnyanb;
            if (virusCheck(VidsotkoveSpivvidnishenyaSpivpaninya, ref count, xRoot, item, AllFoundViruses, endIndexforListBox)) return;
            KilkictbSpivpadinb = 0;
            if (f == 0)
            {
                i++;
                j = 0;
            }
            if (j + 1 == read.Length || i + 1 == VirusListString.Length) return;
        }
    }
  }
}
```

Neural network.cs

```
public class SimpleNeuralNetwork
{
    private NeuralLayerFactory _layerFactory;

    internal List<NeuralLayer> _layers;
    internal double _learningRate;
    internal double[][] _expectedResult;

    public SimpleNeuralNetwork(int numberOfInputNeurons)
    {
    _layers = new List<NeuralLayer>();
    _layerFactory = new NeuralLayerFactory();

    // Create input layer that will collect inputs.
    CreateInputLayer(numberOfInputNeurons);

    _learningRate = 2.95;
}
```

```
public void AddLayer(NeuralLayer newLayer)
{
    if (_layers.Any())
    {
        var lastLayer = _layers.Last();
        newLayer.ConnectLayers(lastLayer);
    }

    _layers.Add(newLayer);
}

public void PushInputValues(double[] inputs)
{
    _layers.First().Neurons.ForEach(x => x.PushValueOnInput
                (inputs[_layers.First().Neurons.IndexOf(x)]));
}

public void PushExpectedValues(double[][] expectedOutputs)
{
    _expectedResult = expectedOutputs;
}

public List<double> GetOutput()
{
    var returnValue = new List<double>();

    _layers.Last().Neurons.ForEach(neuron =>
    {
        returnValue.Add(neuron.CalculateOutput());
    });

    return returnValue;
}
public void Train(double[][] inputs, int numberOfEpochs)
{
    double totalError = 0;

    for(int i = 0; i < numberOfEpochs; i++)
    {
        for(int j = 0; j < inputs.GetLength(0); j ++)
        {
            PushInputValues(inputs[j]);

            var outputs = new List<double>();
```

```
// Get outputs.
_layers.Last().Neurons.ForEach(x =>
{
    outputs.Add(x.CalculateOutput());
});

// Calculate error by summing errors on all output neurons.
totalError = CalculateTotalError(outputs, j);
            HandleOutputLayer(j);
            HandleHiddenLayers();
        }
    }
}

private void CreateInputLayer(int numberOfInputNeurons)
{
    var inputLayer = _layerFactory.CreateNeuralLayer(numberOfInputNeurons,
    new RectifiedActivationFuncion(), new WeightedSumFunction());
    inputLayer.Neurons.ForEach(x => x.AddInputSynapse(0));
    this.AddLayer(inputLayer);
}

/// <summary>
/// Helper function that calculates total error of the neural network.
/// </summary>
private double CalculateTotalError(List<double> outputs, int row)
{
    double totalError = 0;

    outputs.ForEach(output =>
    {
        var error = Math.Pow(output - _expectedResult[row][outputs.IndexOf(output)], 2);
        totalError += error;
    });

private void HandleOutputLayer(int row)
{
    _layers.Last().Neurons.ForEach(neuron =>
    {
        neuron.Inputs.ForEach(connection =>
        {
            var output = neuron.CalculateOutput();
            var netInput = connection.GetOutput();
```

```
var expectedOutput = _expectedResult[row][_layers.Last().Neurons.IndexOf(neuron)];

var nodeDelta = (expectedOutput - output) * output * (1 - output);
var delta = -1 * netInput * nodeDelta;

connection.UpdateWeight(_learningRate, delta);

neuron.PreviousPartialDerivate = nodeDelta;
        });
    });
}


private void HandleHiddenLayers()
{
    for (int k = _layers.Count - 2; k > 0; k--)
    {
        _layers[k].Neurons.ForEach(neuron =>
        {
            neuron.Inputs.ForEach(connection =>
            {
                var output = neuron.CalculateOutput();
                var netInput = connection.GetOutput();
                double sumPartial = 0;

                _layers[k + 1].Neurons
                .ForEach(outputNeuron =>
                {
                    outputNeuron.Inputs.Where(i => i.IsFromNeuron(neuron.Id))
                    .ToList()
                    .ForEach(outConnection =>
                    {
                        sumPartial += outConnection.PreviousWeight *
                                      outputNeuron.PreviousPartialDerivate;
                    });
                });

                var delta = -1 * netInput * sumPartial * output * (1 - output);
                connection.UpdateWeight(_learningRate, delta);
            });
        });
    }
}
```

```
}
```

ІЛЮСТРАТИВНА ЧАСТИНА

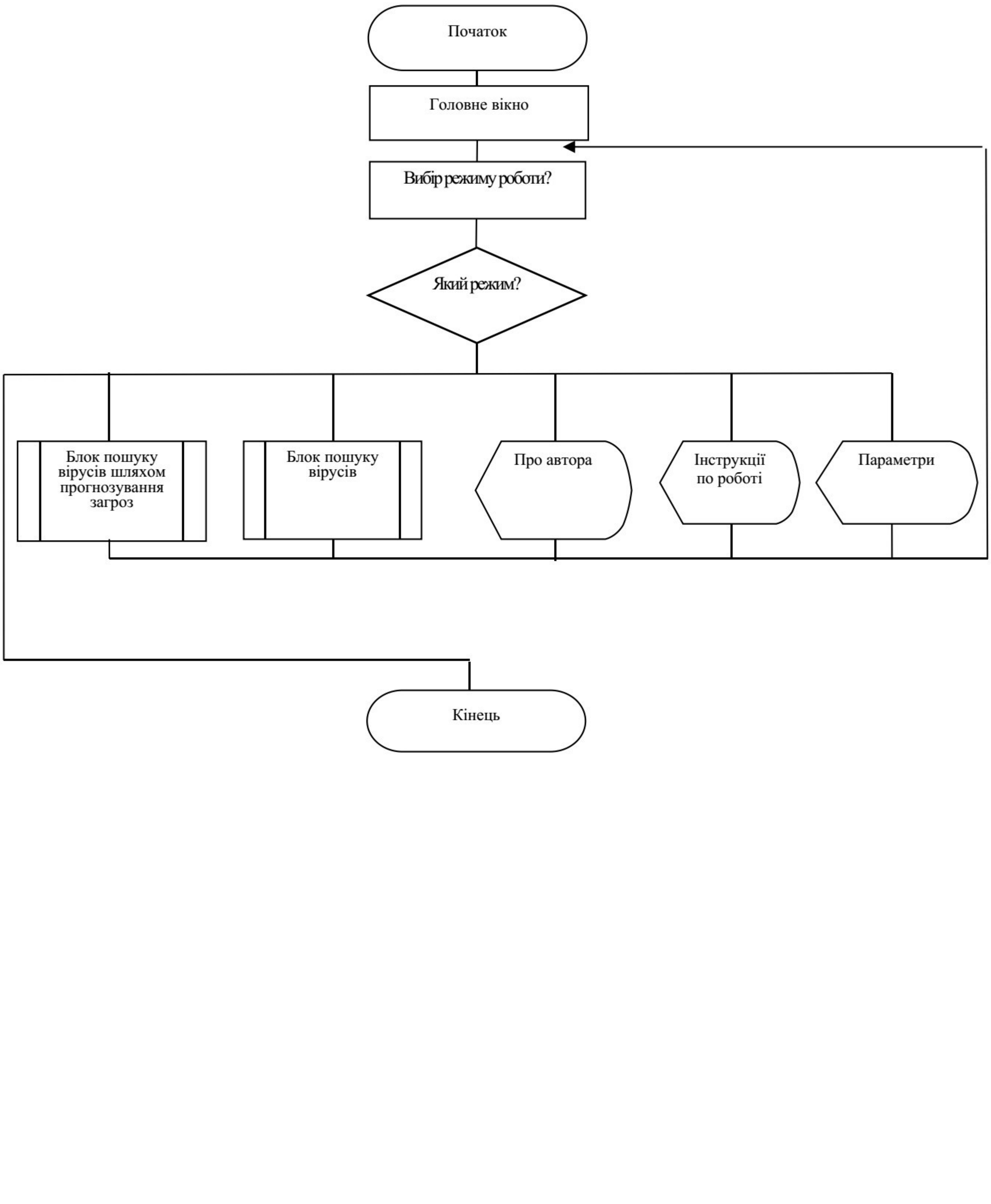
Методи виявлення шкідливого програмного забезпечення

					08-20.МКР.011.00.000 ІЧ1		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Новотарський			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Методи пошуку вірусів	Літ.	Маса / Масштаб
Перевір.		Лукічов В.В					
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м	
Н. Контр.		Лукічов В.В					
Затверд.		ЛужецькийВ.А.					

Загальна схема пошуку вірусів

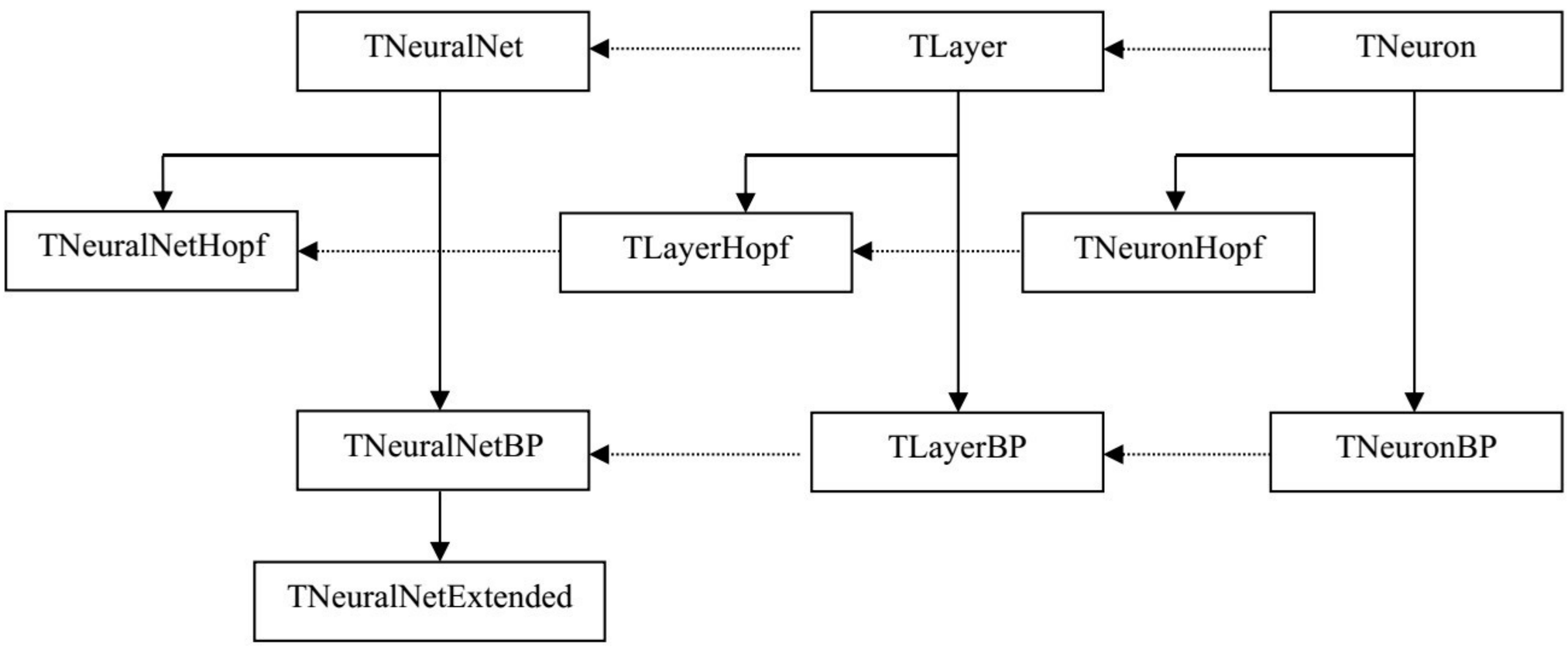
					08-20.МКР.011.00.000 ІЧ2			
Змн	Арк.	№ докум.	Підпис	Дата				
Розроб.		Новотарський			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Загальна схема пошуку вірусів	Літ.	Маса	Масштаб
Перевір.		Лукічов В.В						
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м		
Н. Контр.		Лукічов В.В						
Затверд.		ЛужецькийВ.А.						

Схема функціонування програми



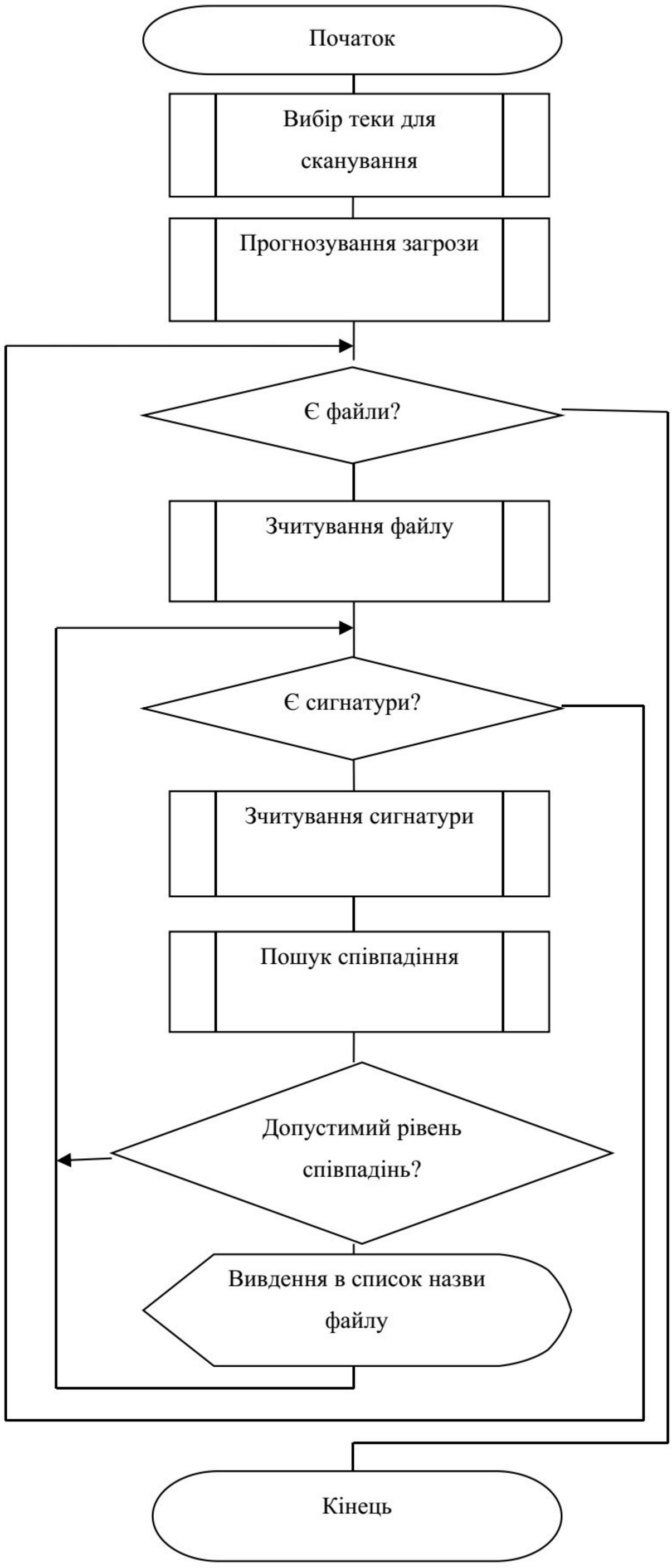
					08-20.МКР.011.00.000 ІЧЗ		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Новотарський			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Схема функціонування програми	Літ.	Маса / Масштаб
Перевір.		Лукічов В.В					
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м	
Н. Контр.		Лукічов В.В					
Затверд.		ЛужецькийВ.А.					

Ієрархія класів нейронної мережі



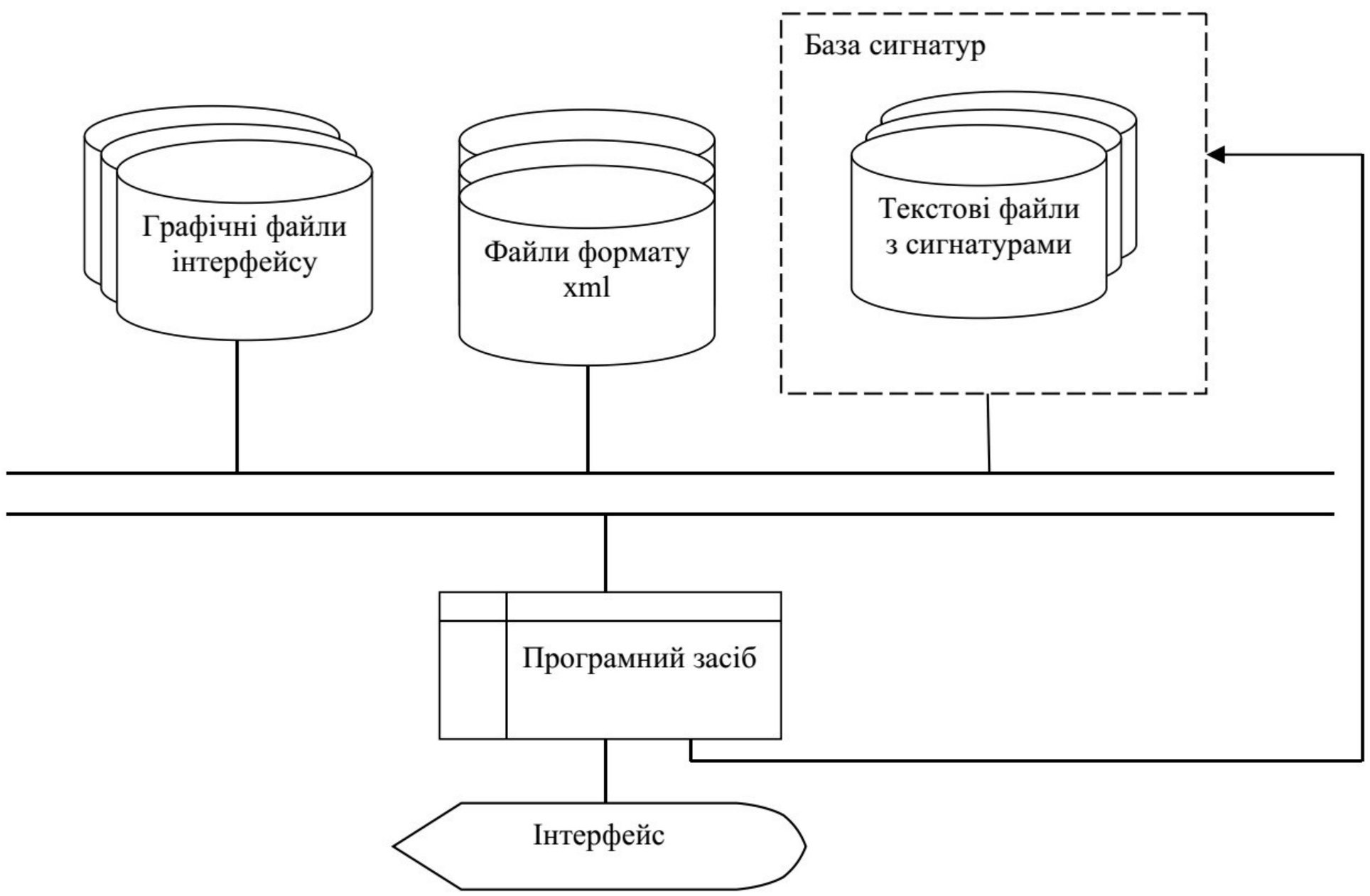
					08-20.МКР.011.00.000 ІЧ4	
Змн	Арк.	№ докум.	Підпис	Дата		
Розроб.		Новотарський І			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Ієрархія класів	Літ. / Маса / Масштаб
Перевір.		Лукічов В.В				
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м
Н. Контр.		Лукічов В.В				
Затверд.		ЛужецькийВ.А.				

Схема алгоритму пошуку шкідливого програмного забезпечення



					08-20.МКР.011.00.000 ІЧ5		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Новотарський			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Схема алгоритму пошуку ШПЗ	Літ.	Маса / Масштаб
Перевір.		Лукічов В.В					
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м	
Н. Контр.		Лукічов В.В					
Затверд.		ЛужецькийВ.А.					

Схема ресурсів програми



					08-20.МКР.011.00.000 ІЧ6		
Змн	Арк.	№ докум.	Підпис	Дата			
Розроб.		Новотарський			Метод пошуку шкідливого програмного забезпечення шляхом прогнозування загроз Схема ресурсів програми	Літ.	Маса / Масштаб
Перевір.		Лукічов В.В					
Реценз.		Азарова А.О.				ВНТУ, гр. 1 БС-18 м	
Н. Контр.		Лукічов В.В					
Затверд.		ЛужецькийВ.А.					