Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи у хмарі для протидії комп’ютерним вірусам”

Виконав здобувач вищої освіти
IV курсу, групи КІ-21-2
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Омельченко С.С.
«____» _____ 2025 р.

Керівник проекту
кандидат фізико-математичних наук, доцент
_____ Петренюк В.І.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Омельченку Станіславу Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи у хмарі для протидії комп’ютерним вірусам*

2. Керівник роботи *Петренюк Володимир Ілліч,канд. фіз.-мат. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 47-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи у хмарі для протидії комп’ютерним вірусам*

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

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Петренюк В.І.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Омельченко С.С.
(прізвище та ініціали)

АНОТАЦІЯ

Омельченко С.С. Програмне забезпечення системи у хмарі для протидії комп'ютерним вірусам. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи у хмарі для протидії комп'ютерним вірусам.

Метою розробки є програмне забезпечення системи у хмарі для протидії комп'ютерним вірусам.

Результат роботи – програмна реалізація системи у хмарі для протидії комп'ютерним вірусам.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: комп'ютерна інженерія, комп'ютерні віруси

ABSTRACT

Omelchenko S.S. Software for a cloud system to combat computer viruses. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed that is intended for a cloud system to combat computer viruses.

The purpose of the development is software for a cloud system to combat computer viruses.

The result of the work is a software implementation of a cloud system to combat computer viruses.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on a PC with Windows 10/11 OS.

The program was developed in the Python environment.

Keywords: computer engineering, computer viruses

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

КМ	–	комп’ютерна мережа
ХСПКВ	–	комплексна система хмарного антивірусного захисту
МЕ	–	міжмережний екран
ПЗ	–	програмне забезпечення
ПК	–	персональний комп’ютер
ACL	–	Access Control List
FTP	–	File Transfer Protocol
http	–	HyperText Transfer Protocol
POP3	–	Post Office Protocol Version 3
SMTP	–	Simple Mail Transfer Protocol
VLAN	–	Virtual Local Area Network

ВСТУП

Актуальність теми. Забезпечити безпеку обчислювальних ресурсів вашого бізнесу не завжди є простим завданням – на щастя, саме тут може допомогти надійний хмарний антивірус. Ці рішення дозволяють налаштувати віртуальний шлюз для всієї вашої мережі, а не мучитися встановленням додатків на кожній вашій бізнес-машині. У результаті ви матимете покращений захист від зловмисного програмного забезпечення та програм-вимагачів, а оскільки це хмарні рішення, у вас не виникне проблем із їх використанням на комп'ютері чи мобільних пристроях.

Найкращі хмарні антивірусні програмні платформи раніше працювали, зіставляючи записи про відомі загрози. Однак тепер вони використовують розширені алгоритми машинного навчання для виявлення небажаної поведінки програмного забезпечення та ізоляції їх, перш ніж вони можуть пошкодити ваші активи. На цьому переваги також не закінчуються. Завдяки хмарному антивірусу вам не потрібно буде постійно оновлювати програмне забезпечення на кількох пристроях

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи у хмарі для протидії комп'ютерним вірусам.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем у хмарі для протидії комп'ютерним вірусам.

– Дослідження системи у хмарі для протидії комп'ютерним вірусам.

– Програмна реалізація системи у хмарі для протидії комп'ютерним вірусам.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі у хмарі для протидії комп'ютерним вірусам.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи у хмарі для протидії комп'ютерним вірусам, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Хмарний антивірус – це рішення безпеки, яке використовує антивірусну хмарну технологію для забезпечення захисту. На відміну від традиційного антивірусного програмного забезпечення, яке покладається на локальні бази даних і обчислювальну потужність, хмарне антивірусне програмне забезпечення перекладає ці завдання на хмарні сервери. Це забезпечує оновлення в режимі реального часу та захист від останніх загроз, не обтяжуючи ресурси вашої системи.

Як працює антивірусна хмарна технологія

Хмарне антивірусне програмне забезпечення працює шляхом сканування файлів і даних через хмарні сервери. Коли здійснюється доступ до файлу, його підпис або поведінка аналізуються в хмарі за допомогою передових алгоритмів і величезних баз даних загроз. Такий підхід забезпечує швидке виявлення та зменшує вплив на продуктивність локальної системи.

Відмінності між традиційним і хмарним антивірусним програмним забезпеченням:

– Використання ресурсів: традиційне антивірусне програмне забезпечення може споживати значні системні ресурси, уповільнюючи роботу пристрою. Хмарний антивірус мінімізує це, обробляючи дані в хмарі.

– Оновлення в реальному часі: Хмарні рішення пропонують миттєві оновлення, забезпечуючи захист від останніх загроз.

– Масштабованість: Хмарне антивірусне програмне забезпечення можна легко масштабувати для захисту кількох пристроїв у різних місцях.

1.2 Область застосування

Роль хмарного антивірусу в хмарних обчисленнях

Інтеграція з хмарними обчислювальними середовищами

Оскільки компанії все більше використовують хмарні обчислення, інтеграція антивірусних рішень хмарних обчислень стає важливою. Антивірусні інструменти для хмарних обчислень захищають дані, що зберігаються та обробляються в хмарі, забезпечуючи безпеку хмарних служб від шкідливих програм та інших кіберзагроз.

Важливість антивірусних рішень для хмарних обчислень

Хмарний антивірусний захист забезпечує масштабовану та гнучку безпеку, адаптуючись до динамічної природи хмарних середовищ. Він пропонує централізоване керування, що спрощує застосування політик безпеки на різних платформах і пристроях, що має вирішальне значення для підтримки надійної безпеки в розподіленій робочій силі.

Як хмарний антивірусний захист виявляє стелс-віруси

Хмарне антивірусне програмне забезпечення чудово справляється з виявленням стелс-вірусів, використовуючи вдосконалені методи виявлення та аналіз даних у реальному часі. Використовуючи потужність хмари, ці рішення можуть виявляти підозрілу поведінку та аномалії, які традиційне антивірусне програмне забезпечення може пропустити, забезпечуючи додатковий рівень захисту від складних загроз.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи у хмарі для протидії комп’ютерним вірусам, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Щоб допомогти вам знайти рішення, яке найкраще підходить вам і вашому бізнесу, я склав список із п’яти найкращих хмарних антивірусних рішень на ринку, хоча варто зазначити, що антивірусна програма є лише одним із аспектів надійної політики IT-безпеки.

Malwarebytes

Переваги:

– Провідний в галузі антивірус.

– Чудово видаляє шкідливі програми.

– Статус безпеки в реальному часі.

– Безкоштовна пробна версія.

Недоліки:

– Інтерфейс користувача може відштовхнути новачків.

– VPN коштує додатково.

Malwarebytes є стандартним рішенням для видалення зловмисного програмного забезпечення з 2006 року, яке працює майже на будь-якій платформі, яку ви можете собі уявити.

Malwarebytes (раніше відомий як Malwarebytes Anti-malware) сканує та видаляє будь-яке шкідливе програмне забезпечення, яке може ховатися у вашій системі, і фактично сканує в пакетних режимах, а не переглядає окремі відкриті файли.

Malwarebytes також включає власне рішення безпеки для захисту кінцевих точок і різноманітні рівні технології виявлення, щоб відповідати багаторівневому

підходу до безпеки. Цей багатовекторний підхід до захисту гарантує, що ваша мережа, пристрої та важливі файли будуть у безпеці.

Хмарна платформа надає Malwarebytes Endpoint Protection через єдиний уніфікований агент кінцевої точки. Це призводить до миттєвого розгортання та керування цією послугою.

Оскільки Malwarebytes Endpoint Protection надається через єдиний уніфікований агент кінцевої точки, ви зможете миттєво розгортати хмарну платформу та керувати нею.

Якщо вам цікаво спробувати Malwarebytes самостійно, ви можете ознайомитися з безкоштовною пробною версією, яка пропонує миттєвий доступ до захисту кінцевих точок (з повною функціональністю до 100 кінцевих точок). Коли ви будете готові зробити рішучий крок, просто зв’яжіться з відділом продажів Malwarebytes, щоб отримати ціну.

Платформа віддаленого керування Avast Business Hub

Переваги:

– Багаторівневий захист.

– Підключіть кілька пристроїв.

– Уніфіковані послуги.

– Єдина панель приладів.

Недоліки:

– Складний інтерфейс.

– Спорадично сповільнюється.

Окрім відомого та потужного безкоштовного антивірусу, Avast також надає спеціальну хмарну платформу безпеки для малого та середнього бізнесу під назвою Avast Business Hub Remote Management Platform. Він забезпечує єдину інформаційну панель керування, з якої можна керувати будь-яким підключеним до нього пристроєм з метою безпеки.

Це може включати будь-що, починаючи від віддаленого доступу для усунення несправностей, передачі файлів або навіть спілкування з

користувачами, на додаток до більш прямих операцій безпеки, таких як захист від вірусів і шкідливих програм. Крім того, Business Hub дозволяє не лише відстежувати загрози безпеці ззовні мережі, а й діяльність у ній.

Фільтруванням веб-вмісту можна керувати через захищений шлюз із захистом електронної пошти та захистом від спаму. Рішення для резервного копіювання та відновлення також включені в стандартну комплектацію.

Пристроями можна керувати в режимі реального часу з широким набором підсумків і звітів. Також вбудовані оповіщення, які можна налаштувати для надсилання через SMS або електронну пошту. Зміни в політиці безпеки, внесені через інформаційну панель, можна негайно застосувати до всіх підключених пристроїв.

Business Hub також тепер може керувати виправленнями в рамках нещодавнього оновлення служби.

Ціна залежить від індивідуальних потреб і розміру бізнесу, але доступні як онлайн-демо, так і безкоштовна пробна версія.

ESET Protect Entry

Переваги:

– 30-денна безкоштовна пробна версія.

– Щит від програм-вимагачів.

– Легке налаштування.

Недоліки:

– Невеликий контроль над кінцевими точками Linux.

ESET – це компанія з безпеки IT, заснована в 1992 році. Її штаб-квартира знаходиться в Братиславі, Словаччина. ESET була визнана найуспішнішою словацькою компанією протягом трьох років поспіль у 2008-2010 роках. Серед їхніх клієнтів Canon, Honda та Greenpeace, які мають десятки тисяч кінцевих точок.

ESET пропонує захист як для домашніх користувачів, так і для компаній. Центр управління безпекою ESET керує продуктами кінцевих точок з «єдиного

блиску» та може бути встановлений на Windows або Linux.

Кінцеві продукти ESET використовують машинне навчання в поєднанні з іншими рівнями захисту. Захист кінцевих точок компанії має здатність виявляти зловмисне програмне забезпечення до виконання, під час виконання та після виконання. Діючи таким чином, ESET може забезпечити надзвичайно високий рівень захисту для ваших пристроїв.

Програмне забезпечення також містить захист від програм-вимагачів, який відстежує всі програми на основі їхніх дій. Це призначено для захисту та блокування процесів, які нагадують поведінку програм-вимагачів.

Існує кілька варіантів, доступних для цього продукту, не в останню чергу хмарне, а також локальне рішення. Крім того, ви можете придбати в партнера, що може бути ідеальним для тих, хто шукає особистої підтримки у своєму рідному штаті.

Як хмарні, так і локальні рішення можна придбати безпосередньо на веб-сайті ESET із планами хмарного захисту 5 пристроїв із можливістю додавання нових онлайн.

Webroot

Просте у використанні антивірусне рішення.

Переваги:

– Потужна консоль управління.

– 30-денна безкоштовна пробна версія.

Недоліки:

– Деякі проблеми з підтримкою.

Webroot – приватна американська компанія, заснована в Боулдері, Колорадо в 1997 році. Вона працює в багатьох країнах, таких як Австралія, Ірландія, Японія та Великобританія. Їхня міжнародна штаб-квартира розташована в Дубліні, Ірландія. Webroot забезпечує безпеку в Інтернеті як для споживачів, так і для компаній.

Webroot використовує багатовекторну систему захисту. Це захищає від

загроз в електронній пошті, веб-браузерах і файлах. Вони використовують «платформу аналізу загроз», яка базується на хмарі. Він класифікує та оцінює понад 95% усіх веб-сайтів тричі на день.

Рішення Endpoint від Webroot пропонує 30-денну безкоштовну пробну версію, яка розгортає захист і сканує за лічені секунди, не потребуючи масових оновлень. Це також дозволяє адміністраторам керувати продуктами захисту Webroot з єдиної консолі керування.

Якщо ви вирішите, що вам потрібно більше від вашого пакету захисту, ви можете придбати рішення, однак Webroot вимагає придбання мінімум 5 кінцевих точок.

Для більш складних потреб ви можете зв’язатися з Webroot, щоб отримати ціну.

Sophos Endpoint Protection

Переваги:

– Автоматичне виявлення вірусів.

– 30-денна безкоштовна пробна версія.

Недоліки:

– Деякі проблеми з оновленням

Sophos – британська компанія, що займається програмним і апаратним забезпеченням безпеки. Вони розробляють і продають антивірусне програмне забезпечення з 1990-х років.

Sophos Endpoint Protection захищає всі ваші пристрої за допомогою однієї спрощеної консолі керування. Це можна встановити на серверах вашої організації або в хмарі.

Sophos автоматизує процес виявлення загроз, дослідження та реагування. Це означає, що час інциденту скорочується. Всі інші тактичні ресурси можна зосередити на їх стратегічному аналізі. Рішення не покладається на сигнатури для виявлення шкідливих програм. Натомість Sophos виявляє загрози ще до того, як вони навіть встигають налаштувати домашню сторінку на вашому пристрої. Це

означає, що продуктивність ваших окремих пристроїв не вплине.

Sophos Endpoint Protection насамперед націлений на підприємства та організації. Щоб отримати пропозицію, вам необхідно заповнити форму запиту. Потім компанія надішле вам цінову пропозицію, яка відповідає вашим потребам. Однак Sophos пропонує 30-денну безкоштовну пробну версію, яка включає автоматичне очищення від зловмисного програмного забезпечення та доступ до хмарного адміністратора та порталу звітності.

Який хмарний антивірус найкращий для вас?

Вирішуючи, який хмарний антивірус використовувати, спершу подумайте про ваші реальні потреби. Бюджетне програмне забезпечення може бути щаднішим для вашого гаманця, але також може надавати відносно обмежений набір інструментів, тому, якщо вам потрібно використовувати розширені параметри, ви можете виявити, що дорожча платформа безпеки набагато більш варта. Крім того, програмне забезпечення вищого класу зазвичай може задовольнити будь-які потреби, тому переконайтеся, що ви добре уявляєте, які функції, на вашу думку, можуть знадобитися від вашого хмарного антивірусу.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Як мова програмування обрана Python. Python – високорівнева мова програмування загального призначення з акцентом на продуктивність розроблювача й читаність коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, у тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне й аспектно-орієнтоване. Основні архітектурні риси – динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточні обчислень і зручні високорівневі структури даних. Код у Python

організовується у функції й класи, які можуть поєднуватися в модулі (які у свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється вільно під дуже ліберальною ліцензією, що дозволяє використовувати його без обмежень у будь-яких застосунках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM і інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови й постановку експериментів над новими можливостями.

Python – мова програмування, що активно розвивається, нові версії (з додаванням/зміною язикових властивостей) виходять приблизно раз у два з половиною року. Внаслідок цього й деяких інших причин на Python відсутні ANSI, ISO або інші офіційні стандарти, їхня роль виконує CPython.

Python портований і працює майже на всіх відомих платформах – від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD і Linux), Plan 9, Mac OS і Mac OS X, iPhone OS 2.0 і вище, Palm OS, OS/2, Amiga, AS/400 і навіть OS/390, Symbian і Android.

При цьому, на відміну від багатьох портуємих систем, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java – Jython, що дозволяє інтерпретаторові виконуватися на будь-якій системі, що підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python й навіть бути написаними на Python. Також кілька проектів забезпечують інтеграцію із платформою Microsoft .NET, основні з яких – IronPython і Python.Net.

Python підтримує динамічну типізацію, тобто тип змінної визначається тільки під час виконання. Тому замість «присвоювання значення змінної» краще говорити про «зв'язування значення з деяким ім'ям». У Python є убудовані типи: бульові, рядки, Unicode-рядки, цілі числа довільної точності, числа із плаваючої

коми, комплексні числа й деякі інші. З колекцій Python підтримує кортежі (*tuples*), списки, словники (асоціативні масиви) і, починаючи з версії 2.4, безлічі. Всі значення в Python є об'єктами, у тому числі функції, методи, модулі, класи.

Додати новий тип можна або написавши клас (class), або визначивши новий тип у модулі розширення (наприклад, написаному мовою С). Система класів підтримує спадкування (одиночне й множинне) і метапрограмування. Можливе спадкування від більшості убудованих типів і типів розширень.

Всі об'єкти діляться на посилальні й атомарні. До атомарного ставляться int, long, complex і деякі інші. При присвоюванні атомарних об'єктів копіюється їхнє значення, у той час як для посилальних копіюється тільки покажчик на об'єкт, таким чином, обидві змінні після присвоювання використовують те саме значення. Посилальні об'єкти бувають змінювані й незмінні. Наприклад, рядки й кортежі є незмінними, а списки, словники й багато інших об'єктів – змінюваними. Кортеж у Python є, по суті, незмінним списком. У багатьох випадках кортежі працюють швидше списків, тому якщо ви не плануєте змінювати послідовність, то краще використовувати саме їх.

Мова має чіткий і послідовний синтаксис, продуману модульність й масштабованість, завдяки чому вихідний код написаних на Python програм легко читаємий.

Python – стабільна й розповсюджена мова. Він використовується в багатьох проектах і в різних якостях: як основна мова програмування або для створення розширень і інтеграції застосунків. На Python реалізоване велика кількість проектів, також він активно використовується для створення прототипів майбутніх програм. Python використовується в багатьох великих компаніях.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне

забезпечення, яке призначено для системи у хмарі для протидії комп’ютерним вірусам.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Переваги хмарного антивірусного захисту

Оновлення в реальному часі та виявлення загроз

Хмарне антивірусне програмне забезпечення забезпечує миттєве оновлення, забезпечуючи захист від найновіших загроз у міру їх появи. Ця можливість виявлення загроз у режимі реального часу має вирішальне значення для захисту від експлойтів нульового дня та швидкого поширення зловмисного програмного забезпечення.

Зменшене використання системних ресурсів

Завдяки обробці даних у хмарі ці рішення зменшують навантаження на локальні пристрої, що сприяє підвищенню продуктивності. Користувачі можуть запускати інтенсивні програми, не відчуваючи сповільнень, які зазвичай пов’язані з традиційним антивірусним скануванням.

Масштабованість і гнучкість

Хмарний антивірусний захист можна легко масштабувати відповідно до зростаючих мереж і адаптувати до мінливих потреб безпеки. Це робить його ідеальним вибором для організацій будь-якого розміру, від малих до великих підприємств.

Хмарний антивірусний захист для бізнесу

Малі та середні підприємства можуть отримати значну користь від хмарного антивірусного захисту.

Важливість:

– Економічно: Зменште витрати на апаратне забезпечення та обслуговування.

– Рішення з можливістю масштабування: легко коригувати в міру зростання бізнесу.

– Покращена безпека: захистіть конфіденційні дані за допомогою вдосконаленого хмарного антивірусного захисту.

Стратегії впровадження:

– Навчання співробітників: переконайтеся, що персонал розуміє важливість кібербезпеки.

– Регулярні аудити: перевіряйте системи на наявність вразливостей.

– Інтеграція з існуючими системами: плавно інтегруйте хмарне антивірусне програмне забезпечення у ваші поточні налаштування.

Кроки для впровадження хмарного антивірусного програмного забезпечення:

1. Оцініть свої потреби: визначте необхідний рівень захисту на основі розміру вашої організації та профілю ризику.

2. Дослідні рішення: досліджуйте різні варіанти хмарного антивірусного програмного забезпечення, щоб знайти найкраще.

3. План розгортання: розробіть план розгортання, який мінімізує збої.

4. Розгорнути та налаштувати: установіть програмне забезпечення та налаштуйте параметри відповідно до ваших політик безпеки.

5. Контролюйте та обслуговуйте: регулярно переглядайте звіти про безпеку та за потреби оновлюйте конфігурації.

Найкращі методи хмарного антивірусного захисту:

– Регулярно оновлюйте програмне забезпечення: переконайтеся, що всі компоненти оновлені для захисту від нових загроз.

– Навчайте користувачів: навчайте персонал передовим методам безпеки, щоб запобігти випадковим порушенням.

– Впроваджуйте багаторівневу безпеку: використовуйте додаткові заходи безпеки, такі як брандмауери, системи виявлення вторгнень і регулярне резервне копіювання.

– Проводьте регулярні аудити: періодично оцінюйте стан безпеки, щоб виявити та усунути вразливі місця.

Вибір правильного хмарного антивірусного програмного забезпечення

Фактори, які слід враховувати:

– Функції безпеки: шукайте захист у режимі реального часу та автоматичні оновлення.

– Сумісність системи: переконайтеся, що програмне забезпечення сумісне з вашими пристроями та операційними системами.

– Вартість: враховуйте свій бюджет і те, чи відповідає безкоштовна версія вашим потребам.

– Підтримка клієнтів: надійна підтримка може мати вирішальне значення для швидкого вирішення проблем.

Поради щодо прийняття обґрунтованого рішення:

– Читайте відгуки: Відгуки користувачів можуть надати цінну інформацію.

– Пробні версії: скористайтеся безкоштовними пробними версіями, щоб протестувати програмне забезпечення.

– Зверніться до експертів: якщо не впевнені, зверніться за порадою до IT-фахівців.

3.2 Розробка структурної схеми

У сучасному цифровому середовищі хмарні обчислення стали невід’ємною частиною бізнес-операцій, пропонуючи неперевершену гнучкість, масштабованість і економічну ефективність. Однак надійні заходи безпеки в хмарі стали обов’язковими, оскільки компанії все більше покладаються на хмарні служби для зберігання та обробки конфіденційних даних. Надійна хмарна безпека гарантує, що додатки та дані залишаються приватними, недоторканими та

доступними на вимогу, захищаючи їх від усіх форм кібератак.

1. Захист конфіденційних даних

Захист конфіденційних даних є однією з фундаментальних причин важливості хмарної безпеки. Організації часто зберігають важливу інформацію в хмарі, включаючи особисті дані клієнтів, фінансові записи та інтелектуальну власність. Без відповідних заходів безпеки ці дані вразливі до злому, несанкціонованого доступу та крадіжки. Застосування надійного шифрування, контролю доступу та постійного моніторингу захищає дані від таких атак.

2. Забезпечення відповідності нормативним вимогам

Багато галузей підпорядковуються суворим нормативним вимогам щодо захисту даних і конфіденційності. Такі нормативні акти, як Загальний регламент захисту даних (GDPR), Закон про перенесення та підзвітність медичного страхування (HIPAA) і Стандарт безпеки даних індустрії платіжних карток (PCI DSS) передбачають спеціальні заходи безпеки для захисту конфіденційної інформації. Надійні методи безпеки в хмарі дозволяють організаціям дотримуватися цих правил, уникаючи юридичних санкцій і зберігаючи довіру клієнтів.

3. Підтримка безперервності бізнесу

Кібератаки, витоки даних або збої в роботі сервісів можуть суттєво порушити бізнес-операції, що призведе до фінансових втрат і репутаційної шкоди. Комплексні заходи безпеки в хмарі, включаючи регулярне резервне копіювання, плани аварійного відновлення та стратегії реагування на інциденти, гарантують, що організації можуть швидко відновлюватися після інцидентів безпеки та підтримувати безперервність бізнесу.

4. Пом’якшення розвитку кіберзагроз

Ландшафт кіберзагроз постійно розвивається, а зловмисники використовують складні методи для використання вразливостей. Хмарні середовища, якщо вони не захищені належним чином, можуть стати цілями для різноманітних атак, у тому числі витоку даних, програм-вимагачів і атак на

відмову в обслуговуванні. Впровадження передових рішень безпеки, таких як багатофакторна автентифікація, системи виявлення вторгнень і регулярні оцінки безпеки, допомагає організаціям випереджати потенційні загрози.

5. Підвищення довіри клієнтів

Клієнти довіряють організаціям свою особисту та фінансову інформацію, очікуючи безпечного поводження з нею. Демонстрація твердої прихильності хмарній безпеці не тільки захищає ці дані, але й підвищує довіру клієнтів. Надійна система безпеки може стати конкурентною перевагою, залучаючи й утримуючи клієнтів, які надають пріоритет захисту даних.

Розуміння моделі спільної відповідальності

У хмарних обчисленнях відповідальність за безпеку розподіляється між постачальником хмарних послуг (CSP) і клієнтом, ця концепція відома як модель спільної відповідальності. У той час як CSP, такі як AWS, Azure і Google Cloud, захищають базову інфраструктуру, клієнти несуть відповідальність за захист своїх даних, програм і доступу користувачів. Неправильне розуміння моделі може призвести до ризиків безпеки, оскільки організації можуть подумати, що CSP відповідає за все, що стосується безпеки. Важливо чітко визначити обов'язки, щоб усе, що стосується безпеки, було належним чином розглянуто.

Основні передові методи безпеки в хмарі

Застосування наведених нижче найкращих практик може значно підвищити рівень безпеки організації в хмарі:

1. Впровадження керування ідентифікацією та доступом (IAM)

Надійний IAM гарантує, що лише призначені користувачі отримують доступ до певних ресурсів. Дотримуючись принципу найменших привілеїв, організації можуть обмежити дозволи користувачів тим, що потрібно для виконання їхніх функцій, зменшуючи площі атак. Регулярний перегляд і оновлення елементів керування доступом дозволяє уникнути випадкового розкриття даних.

2. Увімкніть багатофакторну автентифікацію (MFA)

MFA пропонує другий рівень безпеки, вимагаючи кількох елементів перевірки користувача. Це зменшує ймовірність несанкціонованого доступу через зламані облікові дані. MFA має бути впроваджено в усіх облікових записах користувачів, особливо в облікових записах з високим рівнем привілеїв, як фундаментальний процес у хмарній безпеці.

3. Шифруйте дані в спокої та під час передавання

Шифрування гарантує, що дані залишаються конфіденційними та захищеними від несанкціонованого доступу як під час зберігання (у стані спокою), так і під час передачі (у дорозі). Використання надійних протоколів шифрування та регулярне оновлення ключів шифрування є важливими методами захисту конфіденційної інформації.

4. Регулярно оновлюйте та виправляйте системи

Оновлення систем, програм і інструментів безпеки є життєво важливим для захисту від відомих вразливостей. Регулярне керування виправленнями допомагає запобігти використанню зловмисниками застарілого програмного забезпечення. Автоматизація оновлень, де це можливо, забезпечує своєчасне застосування виправлень.

5. Відстежуйте помилки конфігурації

Неправильні конфігурації є поширеною причиною інцидентів безпеки хмари. Регулярні перевірки та автоматизовані інструменти можуть допомогти виявити та виправити неправильні конфігурації, гарантуючи, що налаштування безпеки відповідають найкращим практикам і політикам організації.

6. Безпечні API

API включені в хмарні служби, але можуть бути точками входу для зловмисників, якщо вони не безпечно розгорнуті. Включення елементів керування автентифікацією, авторизацією та перевіркою введення може допомогти запобігти атакам на API. Важливо також регулярно оновлювати та сканувати API на наявність зловмисної активності.

7. Застосуйте заходи безпеки мережі

Застосовуючи брандмауери, виявлення вторгнень і сегментацію мереж убезпечать хмарні ресурси від незаконного використання та атак. Регулярна перевірка мережевих конфігурацій і контроль доступу дозволять лише законному трафіку проходити до критичних ресурсів

8. Регулярно проводите оцінку вразливості та тестування на проникнення

Регулярне сканування виявляє слабкі місця до того, як ними можуть скористатися зловмисники. Сканування вразливостей дає загальне уявлення про можливі проблеми, тоді як тестування на проникнення надає детальний звіт шляхом імітації реальних атак.

9. Навчайте та навчайте працівників

Людська помилка є суттєвим фактором порушення безпеки. Регулярне навчання гарантує, що співробітники обізнані про політику безпеки, потенційні загрози та найкращі практики. Створення культури обізнаності про безпеку може значно знизити ризик випадкового порушення даних.

10. Розробити та протестувати план реагування на інциденти

Наявність чітко визначеного плану реагування на інциденти дозволяє організаціям швидко й ефективно реагувати на інциденти безпеки. Регулярні тренування та оновлення плану забезпечують готовність і допомагають мінімізувати вплив потенційних порушень.

Sangfor Access Secure (SASE) для покращеної хмарної безпеки

Щоб вирішити нові проблеми безпеки хмари, такі рішення, як Sangfor Access Secure (SASE), пропонують комплексний захист. SASE інтегрує численні функції безпеки в єдину платформу, забезпечуючи безпечний доступ до хмарних ресурсів.

Основні характеристики Sangfor Access Secure:

– Cloud Access Security Broker (CASB): забезпечує дотримання політик безпеки для хмарних програм, гарантуючи, що передача даних відповідає

стандартам організації.

– Доступ до мережі з нульовою довірою (ZTNA) : гарантує, що лише автентифіковані та авторизовані користувачі можуть отримати доступ до певних ресурсів, мінімізуючи ризик бокового переміщення зловмисників.

– Безпечний веб-шлюз (SWG) : захищає від веб-загроз, відстежуючи та фільтруючи інтернет-трафік, запобігаючи доступу до шкідливих сайтів.

– Детальний контроль доступу: дозволяє детально налаштовувати дозволи на основі ідентифікації користувача та контексту, підвищуючи безпеку шляхом обмеження непотрібного доступу.

Впровадження Sangfor Access Secure дає можливість організаціям зміцнити свої позиції в хмарній безпеці шляхом надання інтегрованого та комплексного рішення безпеки.

Ключові причини важливості хмарної безпеки

Оскільки організації продовжують використовувати хмарні обчислення, впровадження надійних заходів безпеки в хмарі має важливе значення для захисту конфіденційних даних, забезпечення дотримання нормативних вимог і пом’якшення нових кіберзагроз. Підприємства можуть посилити свою хмарну безпеку та мінімізувати ризики, запровадивши найкращі практики, такі як багатофакторна автентифікація, шифрування, регулярні оцінки безпеки та навчання співробітників. Крім того, використання передових рішень, таких як Sangfor Access Secure (SASE), може забезпечити комплексний захист від сучасних загроз. Зрештою, проактивний підхід до хмарної безпеки не лише захищає критично важливі активи, але й підвищує довіру клієнтів і стійкість бізнесу у все більш цифровому світі.

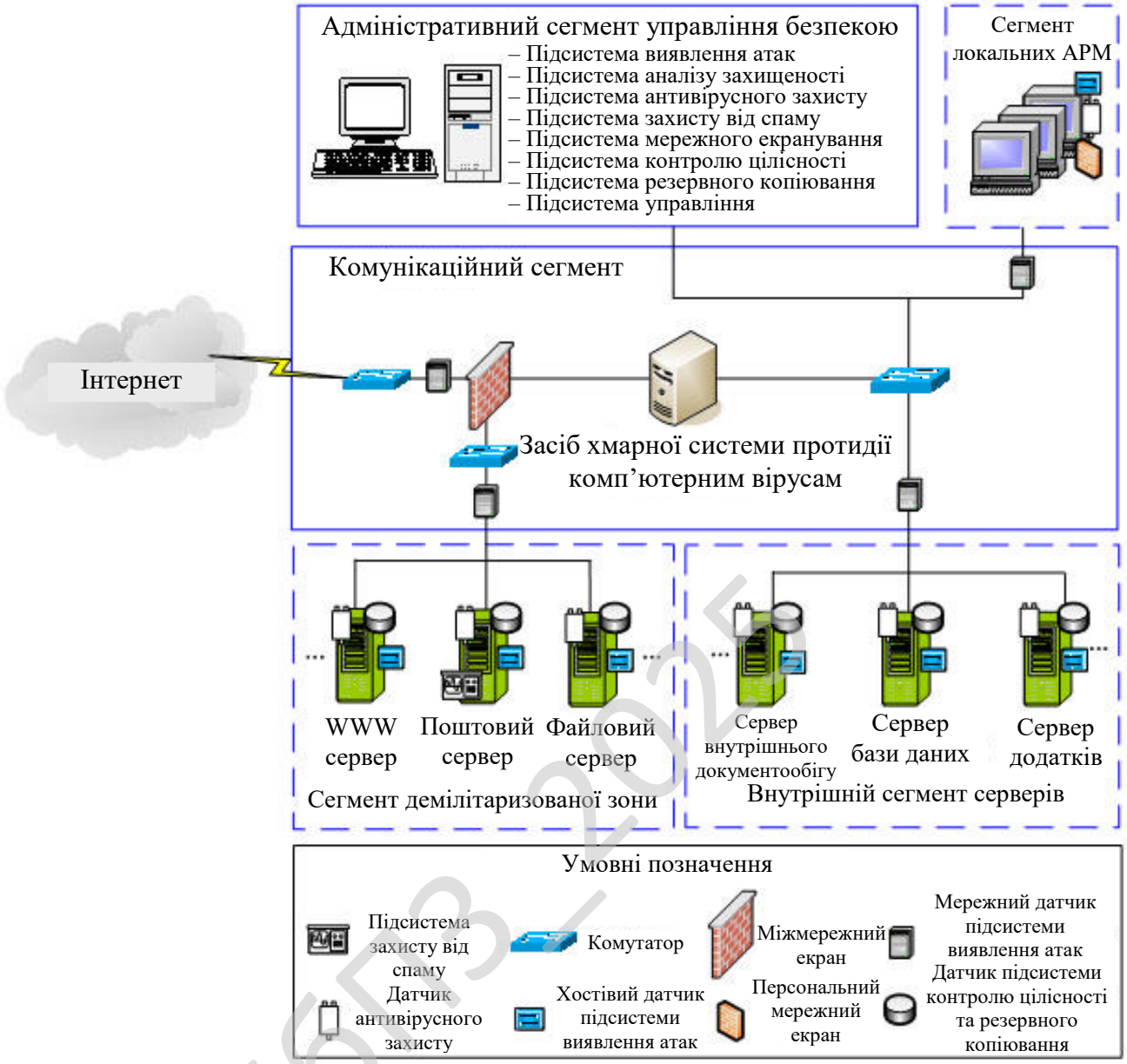

Рисунок 3.1 – Структурна схема системи

3.3 Розробка функціональної схеми

Хмарна система протидії комп'ютерним вірусам, що розроблена у результаті виконання бакалаврської роботи – хмарний захист вашого комп'ютера від шкідливих програм, що включає базові функції забезпечення безпеки вашого ПК. Хмарна система протидії комп'ютерним вірусам використовує новітні технології захисту, завдяки якому забезпечується безпека й стабільна роботу

комп'ютера. Основні функції хмарної системи протидії комп’ютерним вірусам, що розроблений:

– Захист у режимі реального часу.

– Базовий захист при роботі в мережі Інтернет і з електронною поштою.

– Мінімальне завантаження комп'ютера.

– Інтуїтивно зрозумілий інтерфейс.

– Для повноцінного захисту комп'ютера крім хмарної системи протидії комп’ютерним вірусам рекомендується використовувати міжмережний екран.

– Перевірка файлів, веб-сторінок, поштових і ICQ-повідомлень.

– Блокування посилань на заражені веб-сайти й сайти, що перехоплюють інформацію.

– Проактивний захист від невідомих погроз, заснована на аналізі поводження програм.

– Самозахист хмарної системи протидії комп’ютерним вірусам, що розроблений попереджає погрозу вимикання з боку шкідливого ПЗ.

– Система миттєвого виявлення погроз, що моментально блокує нові шкідливі коди.

– Реалізовано модуль «Перевірка посилань», що попереджає про заражені або небезпечних веб-сайти.

– Проактивний захист нового покоління від невідомих погроз.

– Віртуальна клавіатура для безпечного введення логінів, паролів і номерів кредитних карт на веб-сторінках.

– Перевірка операційної системи й установлених програм на наявність уразливостей.

– Налаштування операційної системи й інтернет-браузера для безпечної роботи в мережі Інтернет.

– Відновлення працездатності системи після вірусної атаки.

– Видалення тимчасових файлів інтернет-браузера.

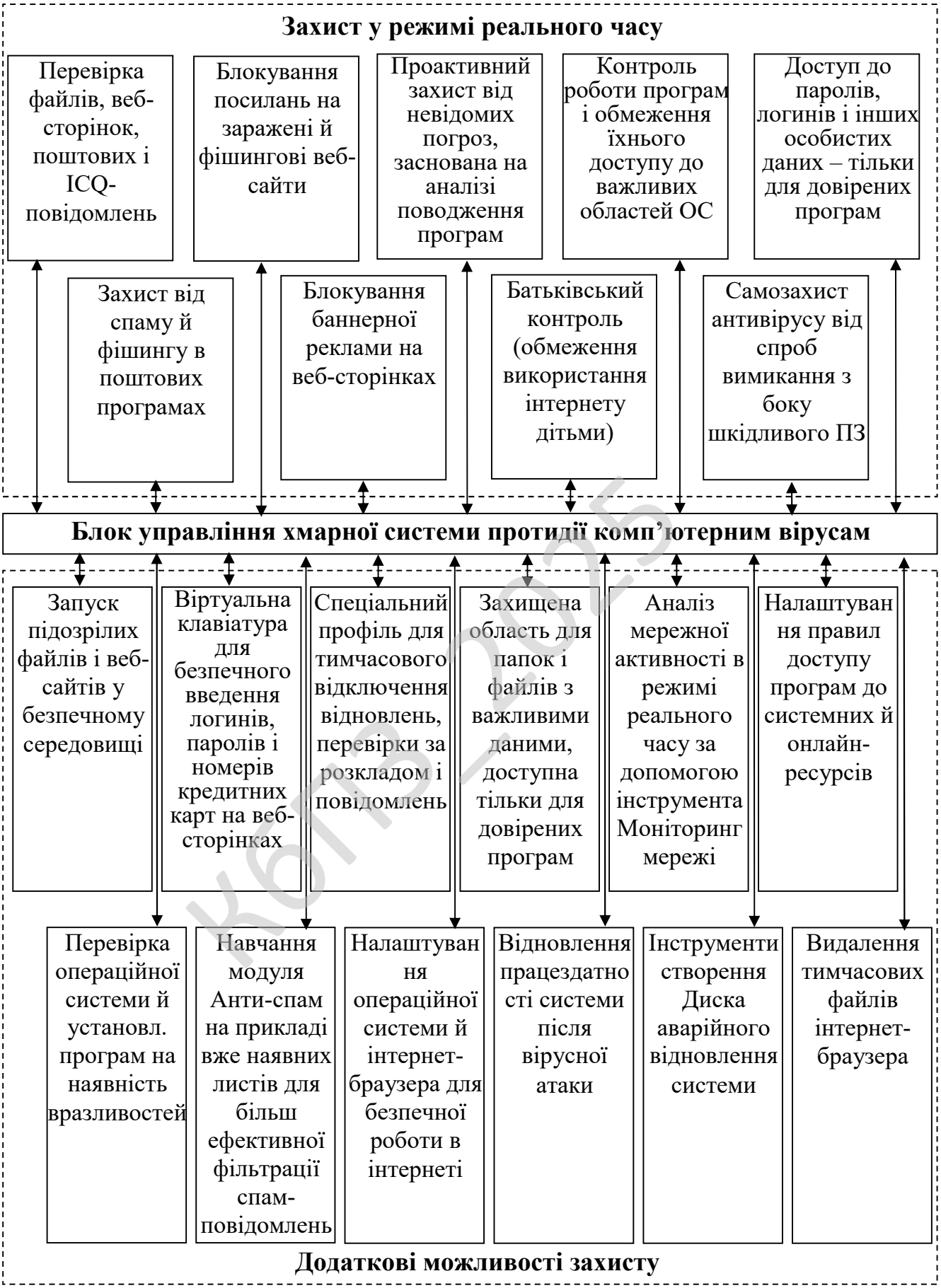

Рисунок 3.2 – Функціональна схема системи

На рисунку 3.2 зображена функціональна схема системи. Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврської роботи, наведена на рисунку 3.3.

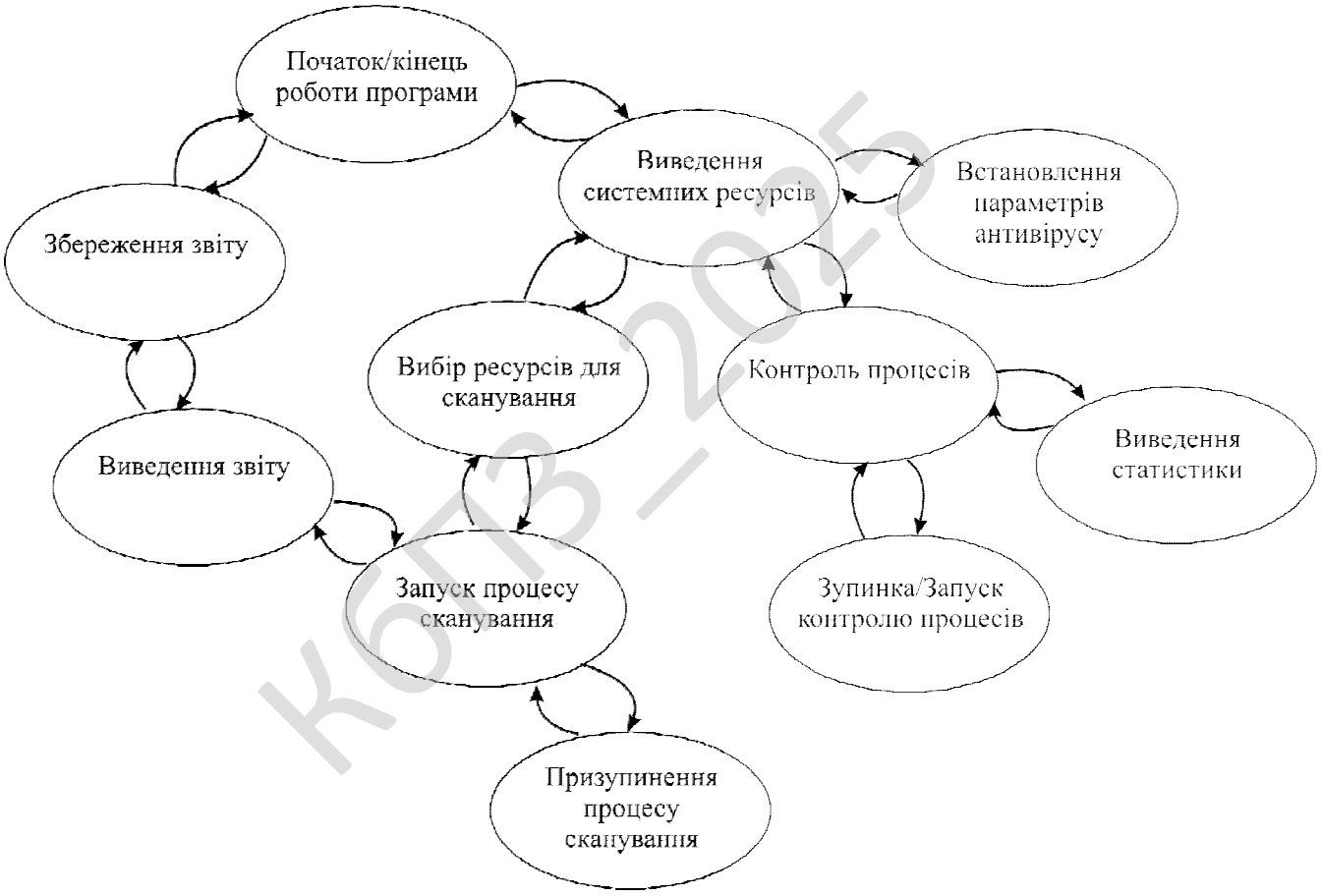

Рисунок 3.3 – Діаграма взаємодії процесів

Першим процесом у розробленій системі являється процес виведення системних ресурсів.

Після нього користувач може перейти до наступних процесів:

– встановлення параметрів хмарної системи протидії комп'ютерним вірусам;

– контроль процесів;

– вибір ресурсів для сканування.

Контроль процесів пов'язаний з наступними процесами:

– виведення статистики;

– зупинки/запуску контролю процесів.

Після вибору ресурсів для сканування слідує процес запуску сканування, що в свою чергу пов'язаний з процесами:

– призупинення сканування;

– виведення звіту;

– збереження звіту.

Таким чином розглянувши структурну схему, функціональну схему та діаграму взаємодії процесів перейдемо до опису блок-схем програмного забезпечення та алгоритмів їх функціонування.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми.

Після цього відбувається виведення логічних дисків, які є у системі.

Далі користувач обирає сканувати йому систему, або ні.

Якщо він вирішує не сканувати, то відбувається перехід до контролю системи.

У іншому випадку, відбувається вибір дисків для сканування.

Після виконання цієї дії, проводиться пошук вірусів на вказаних дисках.

Якщо вірус знаходиться то він знищується, у іншому випадку, відбувається перехід до формування та виведення на екран звіту, про наявність, або ні вірусів у системі, й знищення, або ні цих вірусів.

Крім пошуку вірусів, розроблений у ході виконання бакалаврської роботи програмний продукт дозволяє проводити контроль процесів, які відбуваються у мережі.

Для цього спершу користувач визначається, чи буде він проводити контроль процесів.

Якщо він вирішує його проводити, то відбувається вивід вікна контролю процесів.

У іншому випадку користувач переходить на вікно закінчення роботи з програмним продуктом.

Після виведення вікна контролю процесів, відбувається виведення статистики дій процесів, які функціонують у системі.

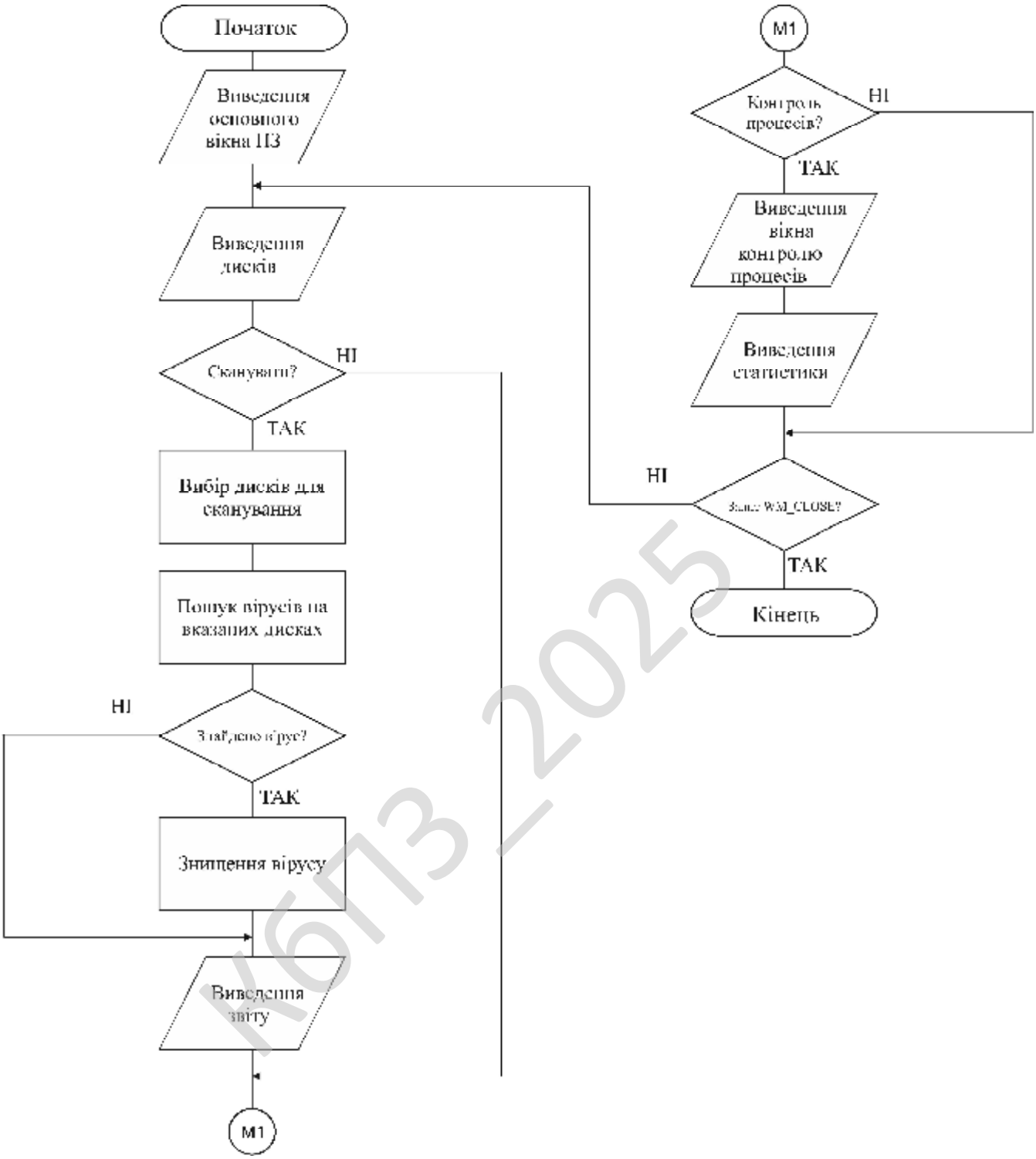

Рисунок 4.1 – Блок-схема роботи основної програми

Після цього користувач обирає працювати йому далі з антивірусом, або завершити роботу з програмою.

Хмарний антивірусний "движок" (Anti-Virus Engine) – це програмний модуль, що призначений для детектування шкідливого програмного забезпечення.

"Движок" є основним компонентом будь-якої антивірусної програми, незалежно від її призначення. Движок використовується як у персональних продуктах – персональний сканер або монітор, так і в серверних рішеннях – сканер для поштового або файлового сервера, мережного екрану або проксі-серверу. Як правило, для детектування шкідливих програм, у більшості "движків" реалізовані наступні технології:

– Статистичний аналіз.

– Евристичний аналіз.

– Емуляція.

– Пошук за "сигнатурами" (унікальній послідовності байт).

– Пошук за контрольними сумами або CRC (контрольної суми з унікальної послідовності байт).

– Використання скороченої маски.

– Криптоаналіз.

Розглянемо кожний із цих методів докладніше.

Статистичний аналіз

Також використовується для детектування поліморфних вірусів. Під час своєї роботи сканер аналізує частоту використання команд процесора, будує таблицю команд, що зустрічаються, процесора, і на основі цієї інформації робить висновок про зараження файлу вірусом. Даний метод ефективний для пошуку деяких поліморфних вірусів, тому що ці віруси використовують обмежений набір команд у декрипторі, тоді як "чисті" файли використовують зовсім інші команди з іншою частотою. Наприклад, всі програми для MS-DOS часто використовують переривання 21h, однак у декрипторі поліморфних DOS-вірусів ця команда практично не зустрічається.

Основний недолік цього методу в тому, що є ряд складних поліморфних вірусів, які використовують майже всі команди процесора й від копії до копії набір використовуваних команд сильно змінюється, тобто за побудованою таблицею частот не представляється можливим виявити вірус.

Евристичний аналіз

Коли кількість вірусів перевищила кілька сотень, антивірусні експерти задумалися над ідеєю детектування шкідливих програм, про існування яких антивірусна програма ще не знає (немає відповідних сигнатур). У результаті були створені так звані евристичні аналізатори. Евристичним аналізатором називається набір підпрограм, які аналізують код файлів, що виконуються, макросів, скриптів, пам'яті або завантажувальних секторів для виявлення в ньому різних типів шкідливих комп'ютерних програм. Існують два принципи роботи аналізатора.

Статичний метод. Пошук загальних коротких сигнатур, які присутні в більшості вірусів (так звані "підозрілі" команди). Наприклад, велика кількість вірусів робить пошук вірусів по масці *.EXE, відкриває знайдений файл, робить запис у відкритий файл. Завдання евристик у цьому випадку – знайти сигнатури, що відбивають ці дії. Потім відбувається аналіз знайдених сигнатур, і, якщо знайдено деяку кількість необхідних і достатніх "підозрілих команд", то приймається рішення про те, що файл інфікований. Великий плюс цього методу – простота реалізації й хороша швидкість роботи, але при цьому рівень виявлення нових шкідливих програм досить низький.

Динамічний метод. Цей метод з'явився одночасно із впровадженням в антивірусні програми емуляції команд процесора (докладніше емулятор описаний нижче). Суть методу полягає в емуляції виконання програми й протоколюванні всіх "підозрілих" дій програми. На основі цього протоколу приймається рішення про можливе зараження програми вірусом. На відміну від статичного методу, динамічний метод більш вимогливий до ресурсів комп'ютера, однак і рівень виявлення в динамічному методі значно вище.

Емуляція

Технологія емуляції коду програм (або Sandboxing) стала відповіддю на появу великої кількості поліморфних вірусів. Ідея цього методу полягає в тому, щоб емулювати виконання програми (як зараженої вірусом, так і "чистої") у спеціальному "оточенні", що називається також буфером емуляції або

"пісочницею". Якщо в емулятор попадає заражений поліморфним вірусом файл, то після емуляції в буфері виявляється розшифроване тіло вірусу, готове до детектування стандартними методами (сигнатурний або CRC пошук).

Сучасні емулятори емулюють не тільки команди процесора, але й виклики операційної системи. Задача написання повноцінного емулятора є досить трудомісткою, не говорячи вже про те, що при використанні емулятора доводиться постійно контролювати дії кожної команди. Це необхідно для того, щоб випадково не виконати деструктивні компоненти алгоритму вірусу.

Слід особливо зазначити, що доводиться саме емулювати роботу інструкцій вірусу, а не трасувати їх, оскільки при трасуванні вірусу занадто велика ймовірність виклику деструктивних інструкцій або кодів, відповідальних за поширення вірусу.

Пошук за "сигнатурами"

Сигнатура – це унікальний "рядок" байт, що однозначно характеризує ту або іншу шкідливу програму. Сигнатурний пошук, у тій або іншій модифікації, використовується для виявлення вірусів та інших шкідливих програм, починаючи з найперших антивірусних програм і дотепер. Незаперечне достоїнство сигнатурного пошуку – швидкість роботи (при використанні спеціально розроблених алгоритмів) і можливості детектування декількох вірусів однією сигнатурою. Недолік – розмір сигнатури для впевненого детектування повинен бути досить великий, як мінімум 8-12 байт (звичайно для точного детектування використовуються набагато більш довгі сигнатури, до 64 байт), отже, розмір антивірусної бази буде досить великим. Крім цього, останнім часом більшу поширеність одержали шкідливі програми, написані на мовах високого рівня (C++, Delphi, Visual Basic), а в таких програм є окремі частини коду, які практично не змінюються (так звана Run Time Library). Неправильно обрана сигнатура неминуче приведе до помилкового спрацьовування – детектування "чистого", не зараженого файлу як зараженого вірусом. Як рішення цієї проблеми пропонується використовувати або дуже великі сигнатури або використовувати

детектування по деяких областях даних, наприклад, таблиці переміщень (relocation table) або текстові рядки, що не завжди добре.

Пошук за контрольними сумами (CRC)

Пошук за контрольними сумами (CRC – cyclic redundancy check), по суті, є модифікацією пошуку за сигнатурами. Метод був розроблений для запобігання основних недоліків сигнатурного пошуку – розміру бази й зменшення ймовірності помилкових спрацьовувань. Суть методу полягає в тому, що для пошуку шкідливого коду береться не тільки "опорний" рядок – сигнатура, а, точніше сказати, контрольна сума цього рядка, але й місце розташування сигнатури в тілі шкідливої програми. Місце розташування використовується для того, щоб не підраховувати контрольні суми для всього файлу. Таким чином, замість 10-12 байт сигнатури (мінімально) використовується 4 байти для зберігання контрольної суми й ще 4 байти – для місця розташування. Однак метод пошуку за контрольними сумами трохи повільніший, ніж пошук за сигнатурами.

Використання масок для виявлення шкідливого коду досить часто буває ускладнений наявністю шифрованого коду (так звані поліморфні віруси), оскільки при цьому або неможливо вибрати маску, або маска максимального розміру не задовольняє умові однозначної ідентифікації вірусу без помилкових спрацьовувань.

Неможливість вибору маски достатнього розміру у випадку поліморфного вірусу легко пояснюється. Шляхом шифрування свого тіла вірус домагається того, що більша частина його коду в ураженому об'єкті є змінною, і, відповідно, не може бути обрана як маска.

Для детектування таких вірусів застосовуються наступні методи: використання скороченої маски, криптоаналіз і статистичний аналіз. Розглянемо ці методи докладніше.

Використання скороченої маски

При поразці об'єктів вірус, що використовує шифрування, перетворить свій код у шифровану послідовність даних:

$$S = F(T),$$

де T – базовий код вірусу;

S – зашифровані коди вірусу;

F – функція шифрування вірусу, що довільно вибирається з деякої множини перетворень {F}.

Спосіб скороченої маски полягає в тому, що вибирається перетворення R зашифрованих кодів вірусу S, таке, що результат перетворення (тобто деяка послідовність даних S') не буде залежати від ключів перетворення F, тобто:

$$S = F(T),$$

$$S' = R(S) = R(F(T)) = R'(T).$$

При застосуванні перетворення R до всіляких варіантів шифрованого коду S результат S' буде постійним при постійному T. Таким чином, ідентифікація уражених об'єктів виконується шляхом вибору S' як скорочена маска й застосування до уражених об'єктів перетворення R.

Криптоаналіз

Цей спосіб полягає в наступному: за відомим базовим кодом вірусу й за відомим зашифрованим кодом (або за "підозріломим" кодом, схожим на зашифроване тіло вірусу) відновлюються ключі й алгоритм програми-розшифровувача. Потім цей алгоритм застосовується до зашифрованої ділянки, результатом чого є розшифроване тіло вірусу. При рішенні цього завдання доводиться мати справу із системою рівнянь.

Як правило, цей спосіб працює значно швидше й займає набагато менше пам'яті, ніж емуляція інструкцій вірусу. Однак рішення подібних систем часто є завданням високої складності.

Причому основна проблема – це математичний аналіз отриманого рівняння або отриманої системи рівнянь. Багато в чому завдання рішення систем рівнянь

при відновленні зашифрованого тіла вірусу нагадує класичне криптографічне завдання відновлення зашифрованого тексту при невідомих ключах. Однак тут це завдання звучить трохи інакше: необхідно з'ясувати, чи є даний зашифрований код результатом застосування деякої відомої з точністю до ключів функції. Причому заздалегідь відомі багато даних для рішення цього завдання: ділянка зашифрованого коду, ділянка незашифрованого коду, можливі варіанти функції перетворення. Більш того, сам алгоритм цього перетворення й ключі також присутні в аналізованих кодах. Однак існує значне обмеження, що полягає в тому, що дане завдання повинне вирішуватися в конкретних границях оперативної пам'яті й процедура рішення не повинна займати багато часу.

База даних хмарного антивірусного "движка"

База даних є невід'ємною частиною хмарного антивірусного "движка". Більш того, якщо вважати що добре спроектований "движок" змінюється не так часто, то антивірусна база змінюється постійно, тому що саме в антивірусній базі перебувають сигнатури, контрольні суми й спеціальні програмні модулі для детектування шкідливих програм. Як відомо, нові віруси, мережні хробаки й інші шкідливі програми з'являються із завидною частотою, і тому дуже важливо, щоб відновлення антивірусної бази відбувалися якнайчастіше. Якщо п'ять років тому було досить щотижневих відновлень, то сьогодні просто необхідно одержувати хоча б щоденні відновлення антивірусної бази.

Також дуже важливо, що саме перебуває в антивірусній базі: чи тільки записи про віруси або ще й додаткові програмні процедури. У другому випадку набагато легше обновляти функціонал хмарного антивірусного "движка" шляхом звичайного відновлення баз.

Підтримка "складних", вкладених об'єктів

За останні кілька років антивірусні "движки" сильно змінилися. Якщо першим антивірусам для того, щоб вважатися першокласною програмою, було досить перевіряти системну пам'ять, файли, що виконуються, й завантажувальні сектори, то вже через кілька років у зв'язку з ростом популярності спеціальних

утиліт упакування виконавчих модулів перед розроблювачами виникло завдання розпакувати упакований файл перед тим, як його сканувати.

Потім нова проблема – віруси навчилися заражати архівні файли (та й самі користувачі найчастіше пересилали заражені файли в архівах). Антивіруси були змушені навчитися обробляти й архівні файли. В 1995 році з'явився перший макровірус, що заражає документи Microsoft Word. Варто помітити, що формат документів, використовуваний Microsoft Word, закритий, і дуже складний. Ряд антивірусних компаній дотепер не вміють повноцінно обробляти такі файли.

Сьогодні, у зв'язку з величезною популярністю електронної пошти, антивірусні "движки" також обробляють і бази поштових повідомлень і самі повідомлення.

Методи детектування

У типовому антивірусному "движку", що реалізований у кожній антивірусній програмі, використовуються всі необхідні технології для виявлення шкідливих програм: ефективний евристичний аналізатор, високопродуктивний емулятор і, що саме головне, грамотна й гнучка архітектура підсистеми детектування шкідливих програм, що дозволяє використовувати всі перераховані вище методи детектування.

Майже в кожному антивірусному "движку" базовим є метод детектування за контрольними сумами. Цей метод був обраний виходячи з вимоги мінімізації розміру антивірусних баз. Однак архітектура "движка" часто настільки гнучка, що дозволяє використовувати кожний з перерахованих вище методів детектирування, що й робиться для деяких особливо складних вірусів. Це дозволяє домогтися високого рівня детектування вірусів. Докладніше архітектура хмарного антивірусного "движка" представлена на схемі далі в тексті.

Практичне застосування способів виявлення поліморфних вірусів (криптоаналіз і статистичний аналіз, застосування скороченої маски й емуляція), зводиться до вибору найбільш оптимального по швидкодії й обсягу необхідної пам'яті методу. Код більшості вірусів, що самошифруються, досить легко

відновлюється процедурою емуляції. Якщо використання емулятора не є оптимальним рішенням, то код вірусу відновлюється за допомогою підпрограми, що реалізує зворотне перетворення – криптоаналіз. Для детектування емуляції вірусів, що не піддаються емуляції і вірусів, для яких не представляється можливим побудувати зворотне перетворення, використовується спосіб побудови скорочених масок.

У деяких, найбільш складних, випадках застосовується комбінація наведених вище способів. Частина коду розшифровувача емулюється, при цьому з розшифровувача виділяються команди, реально відповідальні за алгоритм розшифровки. Потім на основі отриманої інформації будується й вирішується система рівнянь для відновлення коду вірусу і його детектування.

Комбінація способів застосовується також при кратному застосуванні шифрування, коли вірус шифрує своє тіло кілька разів, використовуючи при цьому різні алгоритми шифрування. Комбінований спосіб відновлення інформації або "чиста" емуляція коду розшифровувача часто застосовуються й з тієї причини, що кожний новий вірус повинен бути проаналізований і підключений в антивірусну базу за мінімальні строки, у які далеко не завжди вкладається необхідний математичний аналіз. І в результаті доводиться користуватися більш громіздкими методами детектування вірусу, незважаючи на те, що цілком доцільні методи математичного аналізу алгоритму розшифровувача.

Робота з "складними" об'єктами

Антивірусні "движки" підтримують роботу з величезним числом форматів упакування й архівування. Розроблювачі досить рідко публікують повний (або хоча б досить докладний) перелік підтримуваних форматів. Далі представлена офіційно опублікована інформація про підтримку "складних" форматів в Антивірусі Касперського. В інших антивірусних продуктах список підтримуваних форматів повинен бути приблизно таким же. "Движок" Хмарної системи протидії комп'ютерним вірусам Касперського підтримує роботу з більш ніж 400 різними утилітами впакування файлів, що виконуються, інсталяторів і архиваторов (усього

більше 900 модифікацій, за станом на травень 2003). Серед них пакувальники файлів, що виконуються, і системи шифрування. Самі популярні з них: Diet, AVPACK, COMPACK, Epack, ExeLock, ExePack, Expert, HackStop, Jam, LzExe, LzCom, PaquetBuilder, PGMPAK, PkLite, PackWin, Pksmart, Protect, ProtEXE, RelPack, Rerp, Rjcrush, Rucc, Scramb, SCRNCH, Shrink, Six-2-Four, Syspack, Trap, UCEXE, Univac, UPD, UPX (кілька версій), WWPACK, ASPack (кілька версій), ASProtect (кілька версій), Astrum, BitArts, BJFnt, Cexe, Cheaters, Dialect, DXPack, Gleam, CodeSafe, ELFCrypt, JDPack, JDProtect, INFTool, Krypton, Neolite, ExeLock, NFO, NoodleCrypt, OptLink, PCPEC, PEBundle, PECompact (кілька версій), PCShrink, PE-Crypt, PE-Diminisher, PELock, PEncrypt, PE-Pack (кілька версій), PE-Protect, PE-Shield, Petite, Pex, PKLite32, SuperCede, TeLock, VBox, WWPack32, XLok, Yoda.

Підтримка стількох пакувальників і архіваторов дозволяє скоротити час аналізу нових вірусів, що приводить до збільшення швидкості реакції на появу нового вірусу, і домогтися високого рівня виявлення вже відомих вірусів.

Архіватори й інсталятори (усього більше 60). Самі популярні з них: CAB, ARJ, ZIP, GZIP, Tar, AIN, HA, LHA, RAR, ACE, BZIP2, WiseSFX (кілька версій), CreateInstall, Inno Installer, StarDust Installer, MS Expand, GKWare Setup, SetupFactory, SetupSpecialist, NSIS, Astrum, PCInstall, Effect Office.

Підтримка великої кількості різновидів архіваторів особливо важлива для перевірки поштових систем, тому що гнітюча частина вірусів пересилається поштою в архівіруваному виді. Розпакування об'єктів відбувається незалежно від рівня вкладеності архівів. Наприклад, якщо заражений файл упакований утилітою UPX, а потім файл упакований в архів ZIP, що упакований в архів CAB і т.д., то хмарний антивірусний "движок" однаково повинен змогти дістати вихідний файл і виявити вірус.

Слід зазначити, що подібні міркування носять аж ніяк не теоретичний характер. Так, широко відома троянська програма Backdoor.Rbot, що поширювалася упакованою безліччю різних програм (Ezip, Exe32Pack, ExeStealth,

PecBundle, PECompact, FSG, UPX, Morphine, ASPack, Petite, PE-Pack, PE-Diminisher, PELock, PESpin, TeLock, Molebox, Yoda, Ezip, Krypton і ін.).

Алгоритм розпакування архівів звичайно має достатній інтелект, щоб не розпаковувати всілякі "архівні бомби" – архіви невеликого розміру, у які впаковані величезні файли (з дуже високим ступенем стиску) або кілька однакових файлів. Звичайно для перевірки такого архіву потрібно багато часу, але сучасні антивірусні "движки" часто розпізнають подібні "бомби".

Механізм відновлення антивірусних баз і їхній розмір

Відновлення антивірусних баз звичайно виходять по кілька разів у день. Деякі в стані випускати відновлення раз у годину, деякі – раз в дві години. У кожному разі, при сучасному високому рівні небезпеки в Інтернет таке часте відновлення антивірусних баз цілком виправдано.

Розмір відновлень указує на продуманість архітектури хмарного антивірусного "движка". Так, розмір регулярних відновлень лідируючих у галузі компаній, як правило, не перевищує 30 Кб. При цьому в антивірусні бази звичайно закладене близько 70% функціональності всього хмарного антивірусного "движка". У будь-якому відновленні антивірусної бази може бути додана підтримка нового пакувальника або архіватора. Таким чином, щодня обновляючи антивірусні бази, користувач одержує не тільки нові процедури детектування нових шкідливих програм, але й відновлення всього хмарної системи протидії комп'ютерним вірусам. Це дозволяє дуже гнучко реагувати на ситуацію й гарантувати користувачеві максимальний захист.

Евристичний аналізатор

В евристичному аналізаторі, що входить до складу майже кожного хмарної системи протидії комп'ютерним вірусам, використовуються обидва описані вище методи аналізу – криптоаналіз і статистичний аналіз. Сучасний евристичний аналізатор споконвічно розробляється так, щоб бути розширюваним (на відміну від більшості евристичних аналізаторів першого покоління, які розроблялися для виявлення шкідливих програм тільки у виконуючих модулях).

На сьогоднішній день евристичний аналізатор дозволяє виявляти шкідливі коди у файлах, що виконуються, секторах і пам'яті, а також нові скрипт-віруси й шкідливі програми для Microsoft Office (і інших програм, що використовують VBA), і, нарешті, шкідливий код, написаний на мовах високого рівня, таких як Microsoft Visual Basic.

Гнучка архітектура й комбінація різних методів дозволяє домогтися досить високого рівня детектування нових шкідливих програм. При цьому розроблювачі докладають всі зусилля для того, щоб кількість фіктивних тривог звести до мінімуму. Продукти, що представляються лідерами антивірусної індустрії, надзвичайно рідко помиляються в детектуванні шкідливих кодів.

Схема роботи хмарного антивірусного "движка"

У наведеній нижче схемі описаний зразковий алгоритм роботи хмарного антивірусного "движка". Варто помітити, що емуляція, пошук відомих і невідомих шкідливих програм відбувається одночасно.

Як було сказано вище, під час відновлення антивірусної бази відбувається також відновлення й додавання модулів розпакування упакованих файлів і архівів, евристичного аналізатора й інших модулів хмарного антивірусного "движка".

На рисунку 4.2 зображена блок-схема роботи підпрограми хмарного антивірусного "движка".

Вона працює наступним чином.

Спершу відбувається вибір об’єкту для перевірки.

Якщо це архів, то відбувається розпаковування архіву у тимчасову директорію.

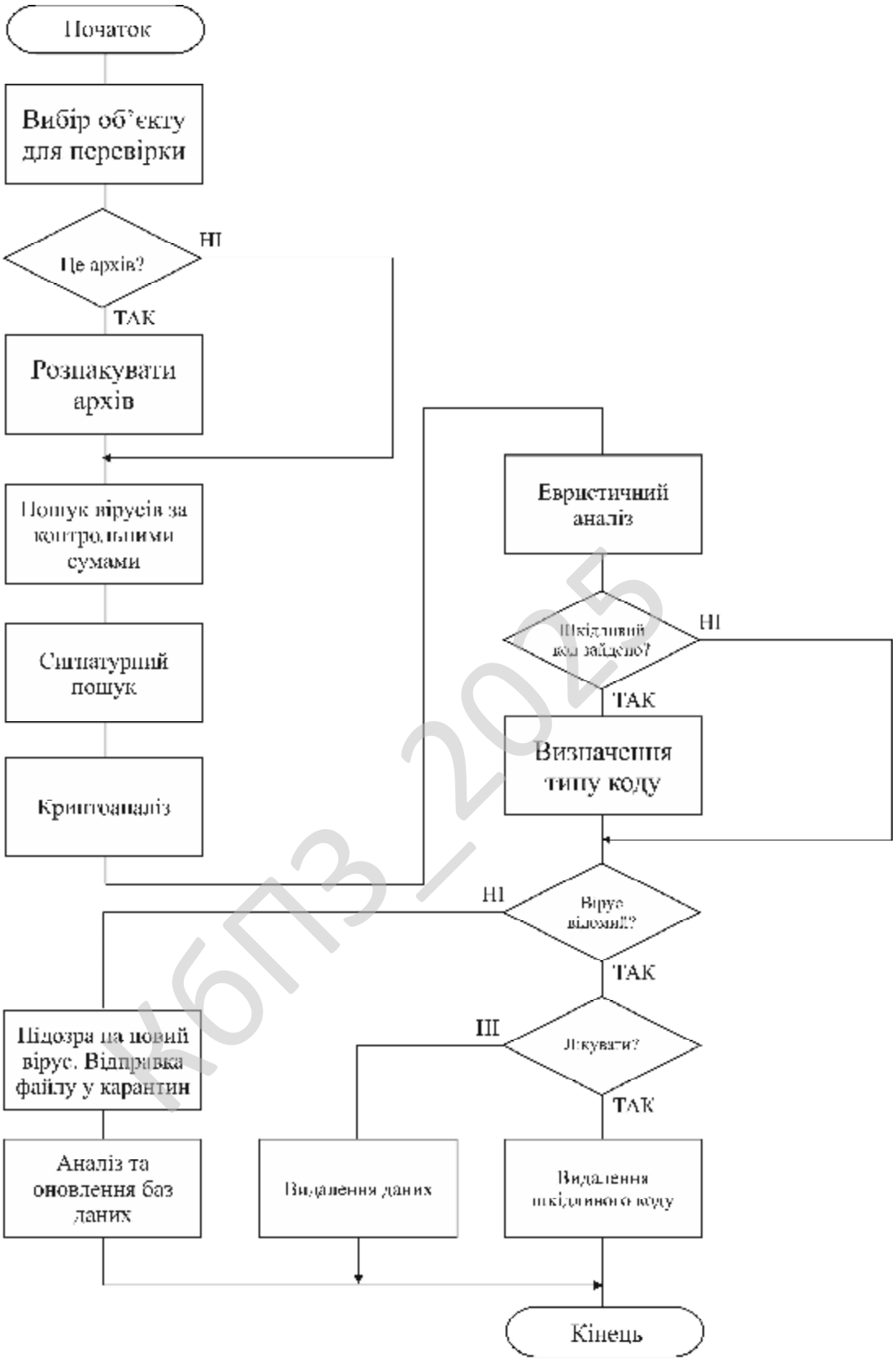

Рисунок 4.2 – Блок-схема роботи підпрограми хмарного антивірусного "движка"

Після цього відбувається пошук вірусів за наступними методами пошуки шкідливого вірусного коду:

– за контрольними сумами;

– сигнатурний пошук;

– криптоаналіз:

– евристичний аналіз.

Якщо шкідливий код знайдено, то виходячи з заданих правил, визначається це старий відомий вірус, або це новий вірус.

Якщо є підозра на новий вірус то проводиться його аналіз, та він додається до бази даних вірусів.

У іншому випадку, якщо вірус вже відомий, то користувач визначає яку дію з ним проводити: лікування або знищення.

Якщо він обирає лікування, то відбувається спроба вилікувати файл, видаливши з нього шкідливий код. Якщо це неможливо, то файл знищується.

Оригінальні технології в антивірусних "движках"

Майже кожний розроблювач антивірусних продуктів реалізує якісь свої технології, що дозволяють зробити роботу програми ефективнішою й більш продуктивною. Деякі із цих технологій мають пряме відношення до пристрою "движка", тому що саме від його роботи часто залежить продуктивність усього рішення. Далі буде розглянутий ряд технологій, що дозволяють значно прискорити перевірку об'єктів і при цьому гарантувати збереження високої якості детектування, а також поліпшити детектування й лікування шкідливого програмного забезпечення в архівних файлах.

Почати треба з технології iChecker. Ця технологія і її аналоги реалізовані майже в кожному сучасному антивірусі. Слід зазначити, що iChecker – назва, запропонована фахівцями "Лабораторії Касперського". Експерти, наприклад, Panda Software називають її UltraFast. Дана технологія дозволяє домогтися розумного балансу між надійністю захисту робочих станцій (і особливо серверів), і використанням системних ресурсів комп'ютера, що захищається. Завдяки цій

технології значно скорочується час завантаження (до 30-40%) операційної системи (у порівнянні із традиційними антивірусними захистами) і час запуску додатків при активному антивірусному захисті. При цьому гарантується, що всі файли на дисках комп'ютера були перевірені й не інфіковані. Основна ідея даної технології – не треба перевіряти те, що не змінювалося, і вже було перевірено. Хмарний антивірусний "движок" веде спеціальну базу даних, у якій зберігаються контрольні суми всіх перевірених (і не інфікованих) файлів. Тепер, перш ніж віддати файл на перевірку, "движок" підраховує й порівнює контрольну суму файлу з даними, що зберігаються в базі даних. Якщо дані збігаються, то це значить, що файл був перевірений і повторна перевірка не потрібно. Варто помітити що час, затрачуваний на підрахунок контрольних сум файлу – значно менше, ніж час антивірусної перевірки.

Особливе місце в роботі хмарної системи протидії комп’ютерним вірусам займає лікування заархівірованих інфікованих об'єктів. iCure – технологія лікування інфікованих файлів в архівах. Завдяки цій технології інфіковані об'єкти усередині архівних файлів будуть успішно вилікувані (або віддалені, залежно від налаштувань хмарної системи протидії комп’ютерним вірусам) без використання зовнішніх утиліт архівації. На сьогоднішній день більшість антивірусів підтримують наступні типи архівів: ARJ, CAB, RAR, ZIP. Завдяки модульній архітектурі й технологіям відновлення хмарного антивірусного "движка" користувач, як правило, може легко обновляти й розширювати список підтримуваних типів архіваторів без перезавантаження хмарної системи протидії комп’ютерним вірусам.

iArc – ще одна технологія роботи з архівними файлами. Ця технологія необхідна для роботи з багатотомними архівами. iArc дозволяє перевіряти багатотомні архіви й виявляти віруси навіть, якщо вони будуть упаковані в багатотомний архів, що, у свою чергу, також буде впакований у багатотомний архів.

Багатопоточність. Хмарний антивірусний "движок" є багатопоточним модулем, і може одночасно обробляти (перевіряти на наявність шкідливих кодів) кілька об'єктів (файли, сектори, скрипти та ін.).

Більшість із перерахованих вище технологій у тому чи іншому виді реалізовано в кожному сучасному антивірусному продукті.

Поліморфні віруси

Протягом всієї статті часто використовувалися терміни "поліморфні" віруси і віруси здатні до самошифрування. Саме цей тип шкідливих кодів вплинув на розвиток антивірусних технологій.

Самошифрування й поліморфічність використовуються практично всіма типами вірусів для того, щоб максимально ускладнити процедуру детектування вірусу. Поліморфні віруси (polymorphic) – це досить важко виявляємі віруси, які не мають сигнатур, тобто, не містять жодної постійної ділянки коду. У більшості випадків два зразки того самого поліморфного вірусу не будуть мати жодного збігу. Це досягається шифруванням основного тіла вірусу й модифікаціями програми-розшифровувача (декриптора). До поліморфних вірусів відносяться ті, детектування яких неможливо (або вкрай важко) здійснити за допомогою так званих вірусних масок – ділянок постійного коду, специфічних для конкретного вірусу. Досягається це двома основними способами – шифруванням основного коду вірусу з непостійним ключем і випадковим набором команд розшифровувача або зміною самого виконуваного коду вірусу. Існують також інші, досить екзотичні приклади поліморфізму: DOS-вірус "Bomber", наприклад, не зашифрований, однак послідовність команд, що передає керування коду вірусу, є повністю поліморфною.

Поліморфізм різного ступеня складності зустрічається у вірусах всіх типів – від завантажувальних і файлових DOS-Вірусів до Windows-вірусів і навіть макро-вірусів.

Поліморфні розшифровувачі

Найпростішим прикладом частково поліморфного розшифровувача є наступний набір команд, у результаті застосування якого жоден байт коду самого вірусу і його розшифровувача не є постійним при зараженні різних файлів:

```
MOV reg_1, count ; reg_1, reg_2, reg_3 вибираються з
MOV reg_2, key ; AX,BX,CX,DX,SI,DI,BP
MOV reg_3, _offset ; count, key, _offset також можуть мінятися
_loop:
xxx byte ptr [reg_3], reg_2 ; xor, add або sub
DEC reg_1
Jxx _loop ; ja або jnc  ; далі слідують зашифровані код і дані вірусу
```

Складні поліморфні віруси використовують значно більш складні алгоритми для генерації коду своїх розшифровувачів: наведені вище інструкції (або їхні еквіваленти) переставляються місцями від зараження до зараження, розбавляються нічого не змінюючими командами, типу NOP, STI, CLI, STC, CLC і т.д.

Повноцінні ж поліморфні віруси використовують ще більш складні алгоритми, у результаті роботи яких у розшифровувачі вірусу можуть зустрітися операції SUB, ADD, XOR, ROR, ROL і інші в довільній кількості й порядку. Завантаження й зміна ключів і інших параметрів шифровки виробляється також довільним набором операцій, у якому можуть зустрітися практично всі інструкції процесора Intel (ADD, SUB, TEST, XOR, OR, SHR, SHL, ROR, MOV, XCHG, JNZ, PUSH, POP ...) з усіма можливими режимами адресації. З'являються також поліморфні віруси, розшифровувач яких використовує інструкції аж до Intel386, а влітку 1997 року виявлений 32-бітний поліморфний вірус, що заражає EXE-Файли Windows 95.

У результаті, на початку файлу, зараженого подібним вірусом, іде набір безглуздих, на перший погляд, інструкцій. Цікаво, що деякі комбінації, які цілком працездатні, не беруться фірмовими дизассемблерами (наприклад, сполучення CS:CS: або CS:NOP). І серед цієї "каші" з команд і даних зрідка прослизають MOV, XOR, LOOP, JMP – інструкції, які дійсно є "робітниками".

Рівні поліморфізму

Існує розподіл поліморфних вірусів на рівні залежно від складності коду, що зустрічається в розшифровувачах цих вірусів. Такий розподіл уперше запропонував доктор Алан Соломон, через якийсь час Весселин Бончев розширив його:

Рівень 1: Віруси, що мають деякий набір розшифровувачів з постійним кодом; при зараженні вибирають один з них. Такі віруси є "напів-поліморфними" і носять також назву "олігоморфних" (oligomorphic). Приклади: "Cheeba", "Slovakia", "Whale".

Рівень 2: Розшифровувач вірусу містить одну або кілька постійних інструкцій, основна ж його частина непостійна.

Рівень 3: Розшифровувач містить невикористовувані інструкції – "сміття" типу NOP, CLI, STI і т.д.

Рівень 4: У розшифровувачі використовуються взаємозамінні інструкції й зміна порядку проходження (перемішування) інструкцій. Алгоритм розшифровки при цьому не змінюється.

Рівень 5: Використовуються всі перераховані вище прийоми, алгоритм розшифровки непостійний, можливо повторне шифрування коду вірусу й навіть часткове шифрування самого коду розшифровувача.

Рівень 6: Permutating-Віруси. Зміні підлягає основний код вірусу – він ділиться на блоки, які при зараженні переставляються в довільному порядку. Вірус при цьому залишається працездатним. Подібні віруси можуть бути незашифрованні.

У наведеній вище класифікації є свої недоліки, оскільки вона виконується за єдиним критерієм – можливість детектувати вірус по коду розшифровувача за допомогою стандартного прийому вірусних масок:

Рівень 1: для детектування вірусу досить мати кілька масок;

Рівень 2: детектування по масці з використанням "wildcards";

Рівень 3: детектування по масці після видалення інструкцій-"сміття";

Рівень 4: маска містить кілька варіантів можливого коду, тобто стає алгоритмічною;

Рівень 5: неможливість детектування вірусу по масці.

Недостатність такого розподілу продемонстрована у вірусі 3-го рівня поліморфізму, що так і називається – "Level3". Цей вірус, будучи одним з найбільш складних поліморфних вірусів, по наведеному вище розподілу попадає у рівень 3, оскільки має постійний алгоритм розшифровки, перед яким стоїть велика кількість команд-"сміття". Однак у цьому вірусі алгоритм генерування "сміття" доведений до досконалості: у коді розшифровувача можуть зустрітися практично всі інструкції процесора i8086.

Якщо зробити розподіл на рівні з погляду антивірусів, що використовують системи автоматичної розшифровки коду вірусу (емулятори), то розподіл на рівні буде залежати від складності емуляції коду вірусу. Можливо детектування вірусу й інших прийомів, наприклад, розшифровка за допомогою елементарних математичних законів і т.д.

Більш об'єктивною буде класифікація, у якій крім критерію вірусних масок беруть участь і інші параметри, наприклад:

– Ступінь складності поліморфного коду (відсоток від всіх інструкцій процесора, які можуть зустрітися в коді розшифровувача).

– Використання спеціальних прийомів, що утрудняють емуляцію антивірусами.

– Сталість алгоритму розшифровувача.

– Сталість довжини розшифровувача.

Зміна виконуваного коду

Найбільш часто подібний спосіб поліморфізму використовується макро-вірусами, які при створенні своїх нових копій випадковим чином міняють імена своїх змінних, вставляють порожні рядки або міняють свій код яким-небудь іншим способом. Таким чином, алгоритм роботи вірусу залишається без змін, але код вірусу практично повністю міняється від зараження до зараження.

Рідше цей спосіб застосовується складними завантажувальними вірусами. Такі віруси впроваджують у завантажувальні сектори лише досить коротку процедуру, що зчитує з диска основний код вірусу й передає на нього керування. Код цієї процедури вибирається з декількох різних варіантів (які також можуть бути розведені "порожніми" командами), команди переставляються між собою й т.д.

Ще рідше цей прийом зустрічається у файлових вірусів – адже їм доводиться повністю міняти свій код, а для цього потрібні досить складні алгоритми. На сьогоднішній день відомі всього два таких віруси, один із яких ("Ply") випадковим чином переміщає свої команди по своєму тілу й заміняє їх на команди JMP або CALL. Інший вірус ("TMC") використовує більш складний спосіб – щоразу при зараженні вірус міняє місцями блоки свого коду й даних, вставляє "сміття", у своїх асемблерних інструкціях установлює нові значення оффсетів на дані, міняє константи й т.д. У результаті, хоча вірус і не шифрує свій код, він є поліморфним вірусом – у коді не присутній постійного набору команд. Більш того, при створенні своїх нових копій вірус міняє свою довжину.

4.2 Захист розробленого програмного забезпечення

Дані у програмному забезпеченні я захищаю за допомогою MISTY1. MISTY1 – блоковий алгоритм шифрування, створений для компанії Mitsubishi Electric криптологом Міцуру Мацуї. Назва є абревіатурою Mitsubishi Improved Security Technology. Алгоритм був розроблений в 1995-1996 рр. Відомі також дві модифікації алгоритму MISTY1: MISTY2 і KASUMI

Шифр став переможцем на Європейському конкурсі NESSIE. У результаті аналізу алгоритму експерти зробили вивід, що ніяких серйозних уразливостей даний алгоритм не має (переважно, завдяки вкладеним мережам Фейстеля, що суттєво утрудняє криптоаналіз). У нього високий запас криптостійкості, алгоритм має високу швидкість шифрування й досить ефективний для апаратної реалізації.

Алгоритм був розроблений на основі теорії «підтвердженої безпеки» проти диференціального й лінійного криптоаналізу. Цей алгоритм був спроектований, щоб протистояти криптоатакам, відомим на момент створення.

З моменту публікації MISTY1 було проведено багато досліджень, щоб оцінити його рівень безпеки.

Диференціальний і неможливий диференціальний криптоаналіз високого порядку ефективно застосовується до блокових шифрів з малим ступенем. Найкращі результати для обох варіантів були отримані для 5-рівневого алгоритму MISTY1 без FL функцій.

Саме FL функції й широкобітні AND/OR операції в сильно утрудняють використання диференціального криптоаналізу, що не заважає проведенню в цьому напрямку всі нових досліджень і досягненню усе більш близьких до розв'язку результатів.

Параметри вихідних даних

MISTY1 – це шифр на основі вкладених мереж Фейстеля з варьюємим числом раундів. Рекомендоване використання 8-раундової версії, але може використовуватися будь-яка кількість раундів, кратне 4-м. Розмір блоку вихідного тексту – 64 біта, розмір ключа – 128 біт.

Для роботи алгоритму також попередньо виконується процедура розширення ключа, яка для 8-мі раундів обчислює 1216 бітів ключової інформації з 128-бітного ключа шифрування.

Структура алгоритму

Для задоволення вимогам конкурсу NESSIE, а також для задоволення завдання мультиплатформеності, в алгоритмі MISTY1 використовувалися наступні методи шифрування:

– Логічні операції.

– Арифметичні операції.

– Операції зрушення.

– Таблиці перестановок.

Як говорилося вище, алгоритм MISTY1 заснований на «вкладених» мережах Фейстеля. Спочатку блок вихідного тексту розбивається на два 32-бітних субблоки, після чого виконується r раундів наступних перетворень[1]:

– У кожному непарному раунді обоє субблоки обробляються операцією FL

– Над оброблюваним субблоком виконується операція FO.

– Результат цих операцій накладається логічною операцією «, що виключає або» (XOR) на неопрацьований субблок.

– Субблоки міняються місцями. Після заключного раунду обоє субблоки ще раз обробляються операцією FL.

Операція FL

Оброблюваний 32-бітний субблок розбивається на два 16-бітних фрагмента, до яких застосовуються операції, де:

– L і R – вхідні значення лівого й правого фрагментів відповідно;

– L' і R' – вихідні значення;

– і – фрагменти j-го підключа i-го раунду для функції FL (процедура розширення ключа докладно описана далі);

– і – побітіві логічні операції «і» і «або» відповідно.

Операція FO

Саме ця функція є вкладеною мережею Фейстеля. Тут, як і раніше, виконується розбивка вхідного значення на два 16-бітних фрагмента, що проходять 3 раунду наступних перетворень:

– На лівий фрагмент операцією XOR накладається фрагмент ключа, де k – номер раунду функції FO.

– Лівий фрагмент обробляється операцією FI.

– На лівий фрагмент накладається операцією XOR значення правого фрагмента.

– Фрагменти міняються місцями.

Після третього раунду операції FO на лівий фрагмент накладається операцією XOR додатковий фрагмент ключа.

Операція FI

Дана операція також представляє собою третій рівень вкладеності мережі Фейстеля. На відміну від двох верхніх рівнів, дана мережа є незбалансованою: оброблюваний 16-бітний фрагмент ділиться на дві частини: 9-бітну ліву й 7-бітну праву. Потім виконуються 3 раунду перетворень, що випливають:

– Ліва частина зазнає обробці S-box.9-бітна частина (в 1-м і 3-м раундах) обробляється таблицею S9, а 7-бітна (в 2-м раунді) – таблицею S7. Дані таблиці описані нижче.

– На ліву частину операцією XOR накладається поточне значення правої частини. При цьому, якщо праворуч 7-бітна частина, вона доповнюється нулями ліворуч, а в 9- бітної частини віддаляються ліворуч два біти.

– У другому раунді на ліву частину операцією XOR накладається фрагмент ключа раунду, а на праву – фрагмент. В інших раундах ці дії не виконуються.

– Ліва й права частини міняються місцями.

Для оптимального розв'язку завдання мультиплатформеності, таблиці S7 і S9 алгоритму MISTY1 можуть бути реалізовані як за допомогою обчислень, так і безпосередньо таблицями.

Розширення ключа

Для 8 раундів алгоритму результатом процедури розширення ключа буде наступний набір ключових значень:

– 20 фрагментів ключа (), кожний з яких має розмір по 16 бітів;

– 32 16-бітних фрагмента ();

– 24 7-бітних фрагмента (при k=4, тобто в 4-м раунді функції FO, операція FI не виконується);

– 24 9-бітних фрагмента.

Виконується дане обчислення в такий спосіб:

1. 128-бітний ключ ділиться на 8 фрагментів … по 16 бітів кожний.

2. Формуються значення: у якості використовується результат обробки значення функцією FI, яка в якості ключа (тобто сукупності необхідних 7- і 9-бітного фрагментів) використовує значення(якщо індекс n фрагмента ключа перевищує 8, то замість нього використовується індекс n-8).

Необхідні фрагменти розширеного ключа «набираються» у міру виконання перетворень із відповідних масивів і згідно з відповідними таблицями

16-бітний фрагмент ділиться на 7-бітний фрагмент і 9-бітний .

Розшифрування

Розшифрування проводиться виконанням тих же операцій, що й при зашифруванні, але з наступними змінами:

– фрагменти розширеного ключа використовуються у зворотній послідовності,

– замість операції FL використовується зворотна їй операція – FLI.

Схеми виконання функції FLI і процедури розшифрування наведено на малюнках 6 і 7 відповідно:

Методи аналізу

Як говорилося на початку розділу, диференціальний і неможливий диференціальний аналізи виявилися ефективні лише до версій шифру з меншою кількістю раундів і без операції FL [2][3]. Проте, на даний момент цей напрямок аналізу, особливе використання слабких ключів, найбільше перспективно, тому що наближене до реальних можливих допущень при використанні алгоритму.

Так само, ученим з Японії був проведений інтегральний аналіз повного алгоритму, використовуючи відкритих текстів зі складністю обчислення, рівної [4].

Лінійний аналіз дав результати тільки для 7-раундової версії шифру, і також без операції FL[5].

Так як MISTY1 створювався, у тому числі, з розрахунку на апаратну реалізацію, має сенс диференціальний аналіз, заснований на використанні атаки по помилках обчислень, що в цьому випадку наближене до реальності.

Висновок

Таким чином, була докладно описана структура алгоритму шифрування MISTY1 і розглянуті методи його аналізу, найбільш прагматичні напрямки дослідження. Далі має бути створення програмної реалізації для більш детального розгляду алгоритму й набір статистичних даних для повного дослідження й пошуку оптимального підходу до аналізу MISTY1.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблене програмне забезпечення реалізує хмарну систему протидії комп’ютерним вірусам.

Дана програма – це простий у використанні і в той же час повноцінний профілактичний антивірусний сканер з високою швидкістю сканування. Сканер має гнучку систему настроювань.

Даний хмарний антивірус забезпечує повноцінний захист комп'ютера від шкідливого ПЗ, а система Контроль процесів – постійно контролює всі процеси користувача, що дозволяє запобігти зараженню системи.

Докладна система звітності, дозволяє перевірити всю інформацію про сканування, і зробити висновки про захищеність системи.

Дана програма виявляє віруси, троянські програми, руткіти та хробаків. Робить пошук і детектування наступних різновидів шкідливого ПЗ:

1. SpyWare, AdvWare програм.
2. Руткітів та інших шкідливих програм.
3. Мережних і поштових хробаків.
4. Троянських програм.

В програму вбудована потужна модульна система, що забезпечує додавання нових можливостей у сканер.

Кожний користувач може створити свій унікальний модуль, що у свою чергу забезпечує максимальну гнучкість сканера.

Для запуску процесу сканування слід вибрати диски, що необхідно перевірити та натиснути кнопку «Сканувати», процес сканування зображено на рисунку 5.1.

На рисунку 5.2 зображено звіт, що виводиться в кінці сканування.

Рисунок 5.1 – Сканування

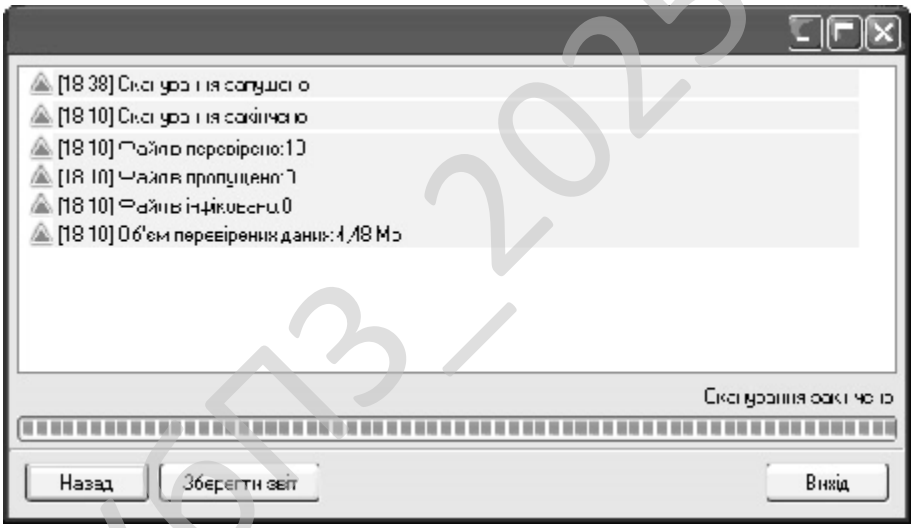

Рисунок 5.2 – Звіт

Для встановлення параметрів програми необхідно натиснути на посилання «Параметри» (рисунки 5.3 – 5.5).

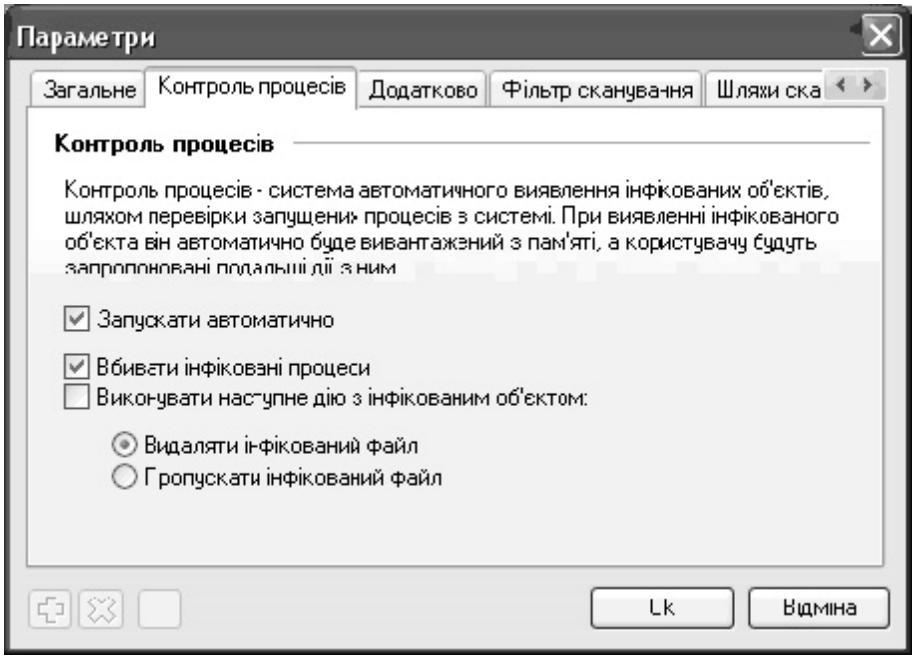

Рисунок 5.3 – Параметри (контроль процесів)

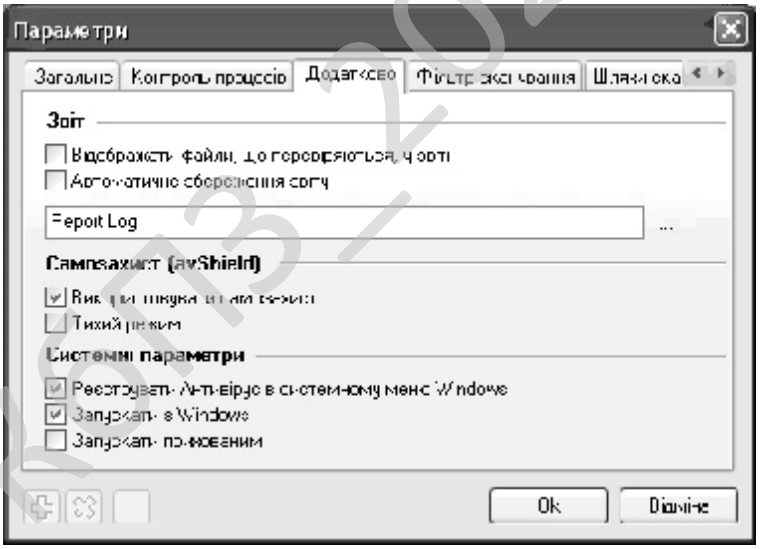

Рисунок 5.4 – Параметри (додатково)

В розробленій програмі є система Контроль процесів, призначена для автоматичного виявлення інфікованих об'єктів, шляхом перевірки запущених процесів в системі.

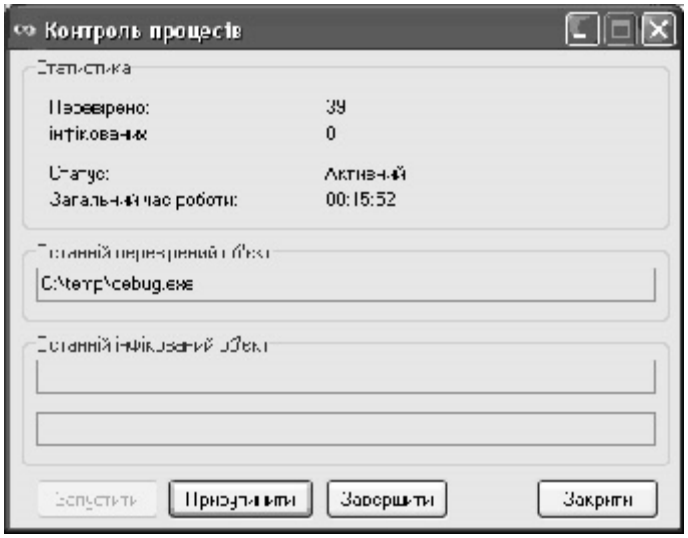

Рисунок 5.5 – Контроль процесів

При виявленні інфікованого об'єкта він автоматично буде вивантажений з пам'яті, а користувачу будуть запропоновані подальші дії з ним. Для перегляду роботи даної системи слід натиснути посилання «Контроль», після чого відкриється вікно, зображене на рисунку 5.6.

Коротку довідку про розроблену програму можна переглянути натиснувши посилання "Про програму…", після чого з'явиться вікно зображене на рисунку 5.5.

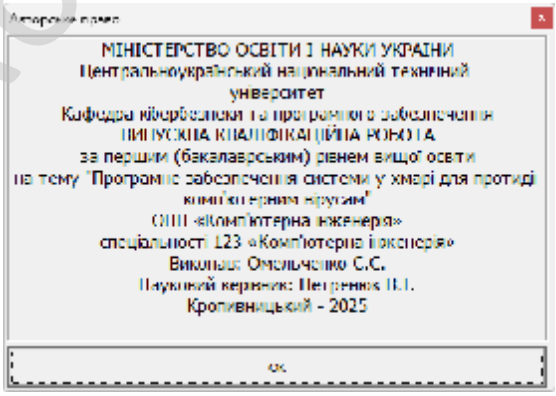

Рисунок 5.6 – Вікно авторського права

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи у хмарі для протидії комп'ютерним вірусам.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем у хмарі для протидії комп'ютерним вірусам.

– Досліджена система у хмарі для протидії комп'ютерним вірусам.

– На основі отриманих результатів досліджень створена програмна реалізація системи у хмарі для протидії комп'ютерним вірусам.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання у хмарі для протидії комп'ютерним вірусам.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об'єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи у хмарі для протидії комп'ютерним вірусам. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм MISTY1.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Awais Rashid, Howard Chivers, George Danezis, Emil Lupu, Andrew Martin. CyBOK The Cyber Security Body of Knowledge. The National Cyber Security Centre. 2019. 854 p.

2. Loren Kohnfelder. Designing Secure Software. No Starch Press. 2022. 332 p.

3. Samir Kumar Rakshit. Ethical Hacker's Penetration Testing Guide. BPB Online. 2022. 509 p.

4. Corey J. Ball. Hacking APIs. No Starch Press. 2022. 353 p.

5. Kevin Beaver. Hacking for Dummies. John Wiley & Sons. 2022. 419 p.

6. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

7. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

8. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

9. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

10. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

11. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

12. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

13. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

14. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

15. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56

16. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

17. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

18. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022,

19. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. Springer, Singapore. pp. 21-34.

20. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

21. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021,* Cracow, Poland, 22-25 September 2021. P. 414-418

22. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) - 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

23. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

24. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

25. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

26. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings*. Volume 2740, 2020, Pages 102-114.

27. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and

cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

28. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

29. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

30. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

31. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

32. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

33. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

34. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and

Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

35. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

36. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43.

37. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660.

38. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

39. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

40. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019.

41. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019.

42. Smirnov, O., Kuznetsov, A., Kiian, A., Gorbenko, Y., Cherep, O., Bexhter L. «Code-based Pseudorandom Generator for the Post-Quantum Period», *2019*

IEEE International Conference on Advanced Trends in Information Theory (IEEE ATIT 2019). 18.12.19-20.12.19 Kyiv Ukraine. P. 204 – 209.

43. Smirnov, O., Kuznetsov, A., Nariezhnii, O., Stelnyk, S., Kokhanovska, T., Kuznetsova T., «Side Channel Attack on a Quantum Random Number Generator», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18 - 21 September 2019. P.713-718.

44. Kuznetsova, T., «Code-Based Schemes for Post-Quantum Digital Signatures», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019;* Metz; France; 18-21 September 2019. P. 707-712.

45. Smirnov, O., Kuznetsov, A., Stefanovych, O., Gorbenko, Y., Krasnobaev, V., Kuznetsova K. «Information Hiding Using 3D-Printing Technology», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.701-706.

46. Smirnov, O., Hu, Z., Vasiliu, Y., Sydorenko, V., Polishchuk, Y., «Abstract Model of Eavesdropper and Overview on Attacks in Quantum Cryptography Systems», *10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2019*; Metz; France; 18-21 September 2019. P.399-405.

47. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019*/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

48. Smirnov, O., Kuznetsov, A., Kiian, A., Babenko, B., Zhosan, H., Prokopovych-Tkachenko, D., «Soft Decoding Method for Turbo-Productive Codes», *2019 3rd International Conference on Advanced Information and Communications Technologies, AICT 2019*, Lviv, Ukraine, 2-6 July, 2019, P. 129-134.

49. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 353-358.

50. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», *2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS)*, Kyiv, Ukraine April 17-19, 2019 P. 347-352.

51. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629.

52. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884.

53. Smirnov A.A., Kuznetsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», *Telecommunications and Radio Engineering*. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

54. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах». *Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка»,* 2024. № 2(26), С. 170–188.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи у хмарі для протидії комп’ютерним вірусам.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 47-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи у хмарі для протидії комп’ютерним вірусам.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи у хмарі для протидії комп’ютерним вірусам;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

- Структурна схема системи – 1 аркуш.
- Функціональна схема системи – 1 аркуш.
- Діаграма процесів – 1 аркуш.
- Блок-схема алгоритму роботи програми – 2 аркуша.
- Пояснювальна записка – 67 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 3.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Петренюк В.І.

Програмне забезпечення системи у хмарі для протидії комп’ютерним вірусам

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 24

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Cloud-based Software System for Countering Computer Viruses
# This Python program implements a comprehensive cloud-based antivirus system.
# The system includes modules for scanning files, updating virus definitions,
# handling API requests, logging, and database operations.

import os
import sys
import time
import random
import threading
import queue
import hashlib
import json
import requests

# Importing Flask for API server functionality
from flask import Flask, request, jsonify

# Global variable for the cloud antivirus system instance
system = None

# ===============
# Logger Class: Responsible for logging system events and messages.
# ===============
class Logger:
    def __init__(self):
        # Initializing the logger and setting the log file name.
        self.log_file = "cloud_antivirus.log"
        self.setup_log()

    def setup_log(self):
        # Setting up the log file by writing an initialization message.
        with open(self.log_file, "a") as file:
            file.write("=== Logger Initialized ===\n")
        # Logger setup complete.

    def log(self, message):
        # Logging a message to the log file with a newline.
        with open(self.log_file, "a") as file:
            file.write(message + "\n")
        # Message logged successfully.

# ===============
# DatabaseHandler Class: Manages storing and retrieving scan logs.
# ===============
class DatabaseHandler:
    def __init__(self):
        # Initializing the database handler and specifying the JSON file for
logs.
        self.db_file = "scan_logs.json"
        self.initialize_db()

    def initialize_db(self):
        # Initializing the database if it does not already exist.
        if not os.path.exists(self.db_file):
            with open(self.db_file, "w") as db:
                json.dump({"scans": []}, db)
        # Database initialization complete.

    def save_scan_result(self, result):
        # Saving the scan result to the database file.
        try:
            with open(self.db_file, "r") as db:
                data = json.load(db)
        except Exception as e:
```

```
            data = {"scans": []}
        data["scans"].append(result)
        with open(self.db_file, "w") as db:
            json.dump(data, db, indent=4)
        # Scan result saved successfully.

# ===============
# VirusScanner Class: Provides functionality to scan files and update virus
definitions.
# ===============
class VirusScanner:
    def __init__(self):
        # Initializing the virus scanner with default virus definitions.
        self.virus_definitions = {}
        self.load_default_definitions()

    def load_default_definitions(self):
        # Loading default virus definitions into the system.
        self.virus_definitions = {
            "eicar_test": "44d88612fea8a8f36de82e1278abb02f",
            " _virus": "abcdef1234567890abcdef1234567890"
        }
        # Default virus definitions loaded.

    def scan_file(self, file_path):
        # Scanning the file at the given path for viruses.
        if not os.path.exists(file_path):
            # File does not exist; returning error.
            return {"status": "error", "message": "File does not exist"}
        try:
            # Opening the file in binary mode.
            with open(file_path, "rb") as f:
                file_data = f.read()
            # Calculating the MD5 hash of the file data.
            file_hash = hashlib.md5(file_data).hexdigest()
            # Iterating over virus definitions to check for a match.
            for virus_name, virus_signature in self.virus_definitions.items():
                # Checking if the virus signature exists in the file hash.
                if virus_signature in file_hash:
                    # Virus detected; returning infected status.
                    return {"status": "infected", "virus": virus_name}
            # No virus signature matched; file is clean.
            return {"status": "clean"}
        except Exception as e:
            # An error occurred during file scanning.
            return {"status": "error", "message": str(e)}

    def update_definitions(self, new_definitions):
        # Updating the virus definitions with new data.
        self.virus_definitions.update(new_definitions)
        # Virus definitions updated successfully.

# ===============
# CloudAPI Class: Implements API endpoints for scanning and updating via HTTP
requests.
# ===============
class CloudAPI:
    def __init__(self, antivirus_system):
        # Initializing the CloudAPI with a reference to the antivirus system.
        self.antivirus_system = antivirus_system
        self.app = Flask(__name__)
        self.setup_routes()

    def setup_routes(self):
      # Setting up API routes for scanning, updating definitions,
      #  and checking status.
        @self.app.route("/scan", methods=["POST"])
        def api_scan():
```

```
            # API endpoint to scan a file.
            data = request.get_json()
            if "file_path" not in data:
                # File path not provided in the request.
                return jsonify({"status": "error", "message": "No file
provided"}), 400

            # Scanning the file using the system's scanner.
            result = self.antivirus_system.scanner.scan_file(data["file_path"])
            # Saving the scan result to the database.
            self.antivirus_system.db_handler.save_scan_result(result)
            # Logging the scan event.
            self.antivirus_system.logger.log("File scanned via API: " +
data["file_path"])

            # Returning the scan result.
            return jsonify(result)

        @self.app.route("/update_definitions", methods=["POST"])
        def api_update_definitions():
            # API endpoint to update virus definitions.
            data = request.get_json()
            if "definitions" not in data:
                # Virus definitions not provided in the request.
                return jsonify({"status": "error", "message": "No definitions
provided"}), 400
            # Updating virus definitions using the provided data.

self.antivirus_system.scanner.update_definitions(data["definitions"])
            # Logging the update event.
            self.antivirus_system.logger.log("Virus definitions updated via
API")
            # Returning a success message.
            return jsonify({"status": "success", "message": "Definitions
updated"})

        @self.app.route("/status", methods=["GET"])
        def api_status():
            # API endpoint to check the system status.
            return jsonify({"status": "running"})

    def run(self, host="0.0.0.0", port=5000):
        # Running the Flask API server.
        self.antivirus_system.logger.log("Starting Flask API server on
{}:{}".format(host, port))
        self.app.run(host=host, port=port, threaded=True)

# ===============
# CloudAntivirusSystem Class: Integrates all components to form the complete
# system.
# ===============
class CloudAntivirusSystem:
    def __init__(self):
        # Initializing the cloud antivirus system components.
        self.logger = Logger()
        self.db_handler = DatabaseHandler()
        self.scanner = VirusScanner()
        self.task_queue = queue.Queue()
        self.workers = []
        self.running = True
        self.initialize_workers()

    def initialize_workers(self):
        # Initializing worker threads for scanning tasks.
        number_of_workers = 5
        for i in range(number_of_workers):
            worker = threading.Thread(target=self.worker_function)
            worker.daemon = True
            worker.start()
```

```
            self.workers.append(worker)
            self.logger.log("Worker thread {} started".format(i))
        # All worker threads initialized.

    def worker_function(self):
        # Worker thread function that processes scanning tasks.
        while self.running:
            try:
                # Attempting to get a task from the queue.
                task = self.task_queue.get(timeout=1)
                # Processing the scanning task.
                result = self.scanner.scan_file(task["file_path"])
                # Saving the scan result to the database.
                self.db_handler.save_scan_result(result)
                # Logging the completed scan task.
                self.logger.log("Scanned file: " + task["file_path"])
                # Marking the task as done.
                self.task_queue.task_done()
            except queue.Empty:
                # No task in the queue; continue looping.
                continue

    def add_scan_task(self, file_path):
        # Adding a new scanning task to the task queue.
        task = {"file_path": file_path}
        self.task_queue.put(task)
        # Logging the addition of a new scan task.
        self.logger.log("Added scan task for file: " + file_path)

    def shutdown(self):
        # Shutting down the cloud antivirus system.
        self.running = False
        # Logging the shutdown process.
        self.logger.log("Shutting down workers")
        for worker in self.workers:
            worker.join()
        # All workers have been shut down.
        self.logger.log("All workers have been shut down")

# ===============
# Function: query_update_server
# Queries a remote server for updated virus definitions.
# ===============
def query_update_server():
    # Defining the URL for the virus definitions update server.
    url = "https://example.com/api/virus_definitions"
    try:
        # Making an HTTP GET request to the update server.
        response = requests.get(url, timeout=5)
        if response.status_code == 200:
            # Parsing JSON response for new definitions.
            new_definitions = response.json()
            # Returning the new virus definitions.
            return new_definitions
        else:
            # Non-200 response received; returning an empty dictionary.
            return {}
    except Exception as e:
        # Exception occurred during the HTTP request.
        return {}

# ===============
# Function: _file_upload
# A client uploading a file for scanning.
# ===============
def _file_upload(file_path):
    # Adding the file to the system's scan task queue.
    # global system
```

```
    if system is not None:
        system.add_scan_task(file_path)
    else:
        # System not initialized; printing error message.
        print("Antivirus system is not initialized.")

# ===============
# Function One: A function to extend code lines.
# ===============
def _function_one():
    # Yaar, this is function one for extra verbosity.
    for i in range(10):
        # Iteration start in _function_one.
        print("Function One, iteration:", i)
        # Iteration end in _function_one.
    # End of _function_one.
    return None

# ===============
#  Function Two: Another  function for code expansion.
# ===============
def _function_two():
    # Bhai, starting  function two.
    total = 0
    for j in range(20):
        # Processing iteration in _function_two.
        total += j * random.randint(1, 10)
        # Intermediate value updated in _function_two.
    print(" Function Two total is:", total)
    # Completed  function two.
    return total

# ===============
#  Function Three: Yet another function with additional steps.
# ===============
def _function_three():
    # Starting  function three with extra steps.
    sample_list = []
    for k in range(15):
        # Appending values in _function_three.
        sample_list.append(k ** 2)
        # Value appended in _function_three.
    # Printing the sample list in _function_three.
    print(" Function Three sample list:", sample_list)
    # End of  function three.
    return sample_list

# ===============
#  Function Four: Additional  function for extended code lines.
# ===============
def _function_four():
    # Commencing  function four execution.
    result_dict = {}
    for m in range(5):
        # Creating key-value pairs in _function_four.
        result_dict["key_" + str(m)] = m * 100
        # Key-value pair added in _function_four.
    # Displaying the result dictionary from _function_four.
    print(" Function Four result:", result_dict)
    #  function four execution complete.
    return result_dict

# ===============
# Function Five
# ===============
def _function_five():
    # Initiating  function five.
    counter = 0
    while counter < 10:
```

```
        # Loop iteration in _function_five.
        print(" Function Five counter at:", counter)
        counter += 1
        # Counter incremented in _function_five.
    #  function five completed.
    return counter

# ===============
# Function: run__functions
# Runs all  functions sequentially.
# ===============
def run__functions():
    # Running _function_one.
    _function_one()
    # Running _function_two.
    _function_two()
    # Running _function_three.
    _function_three()
    # Running _function_four.
    _function_four()
    # Running _function_five.
    _function_five()
    # All  functions executed.

# ===============
# Function: periodic_update
# Periodically queries the update server for virus definitions.
# ===============
def periodic_update(interval_seconds=60):
    # Starting the periodic update loop.
    while True:
        # Querying the update server for new definitions.
        new_defs = query_update_server()
        if new_defs:
            # If new definitions are received, update the scanner definitions.
            system.scanner.update_definitions(new_defs)
            # Logging the update event.
            system.logger.log("Virus definitions updated via periodic update.")
        else:

        # No new definitions received during this cycle.
        system.logger.log("Periodic update: No new virus definitions found.")

        # Sleeping for the specified interval before next update.
        time.sleep(interval_seconds)

       # End of periodic update loop.

# ===============
# Main Function: Entry point for the cloud antivirus system.
# ===============
def main():
    # Initializing the global antivirus system.
    global system
    system = CloudAntivirusSystem()
    # Logging the initialization of the antivirus system.
    system.logger.log("Cloud Antivirus System initialized successfully.")

    # Starting a background thread for periodic virus definition updates.
    update_thread = threading.Thread(target=periodic_update, args=(120,))
    update_thread.daemon = True
    update_thread.start()
    system.logger.log("Periodic update thread started.")

    # Running  functions to additional processing.
    run__functions()
    system.logger.log(" functions executed.")

    # Creating an instance of the CloudAPI to serve HTTP requests.
```

```
    api = CloudAPI(system)
    system.logger.log("CloudAPI instance created. Ready to serve API requests.")

    # Starting the Flask API server.
    try:
        api.run(host="0.0.0.0", port=5000)
    except KeyboardInterrupt:
        # Handling keyboard interrupt for graceful shutdown.
        system.logger.log("KeyboardInterrupt received. Initiating shutdown
sequence.")
        system.shutdown()
        sys.exit(0)

# ===============
# Ensuring that main() runs when the script is executed directly.
# ===============
if __name__ == "__main__":
    main()
```

Файл Ext1.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Extended Cloud Antivirus System Extensions Implementation 1
# The following extensions are implemented:
# 1. Distributed Orchestration Module for Scanning Tasks
# 2. Cloud Sandbox Environment Manager Module
# 3. Forensic Analysis Module
# 4. Benchmarking and Performance Testing Module
# 5. Synthetic Data Generator Module

import threading
import time
import random
import string
import hashlib
import json
import datetime
import queue
import math

# ===============
# Extension 1: Distributed Orchestration Module for Scanning Tasks
# ===============
class DistributedOrchestrator:
    # Конструктор класу DistributedOrchestrator
    def __init__(self):
        # Ініціалізація черги завдань для розподіленого сканування
        self.task_queue = queue.Queue()
        # Ініціалізація списку результатів сканування
        self.results = []
        # Ініціалізація блокування для синхронізації доступу до результатів
        self.lock = threading.Lock()
        # Ініціалізація списку симульованих вузлів
        self.nodes = []
        # Прапорець для контролю виконання вузлів
        self.running = True
        # Виклик функції ініціалізації вузлів
        self.initialize_nodes()

    def initialize_nodes(self):
        # Створення 5 симульованих вузлів (threads) для обробки завдань
        for i in range(5):
            node_thread = threading.Thread(target=self.node_worker, args=(i,))
            node_thread.daemon = True
            node_thread.start()
            self.nodes.append(node_thread)
            # Логування запуску вузла
            print("DistributedOrchestrator: Node {} initialized.".format(i))
        # Завершення ініціалізації всіх вузлів.

    def node_worker(self, node_id):
        # Функція робітника вузла, що обробляє завдання з черги
        while self.running:
            try:
                # Отримання завдання з черги з таймаутом
                task = self.task_queue.get(timeout=2)
                # Логування початку обробки завдання
                print("Node {}: Processing task '{}'.".format(node_id, task))
                # Виклик функції обробки завдання
                result = self.process_task(task, node_id)
                # Додавання результату в загальний список з блокуванням
                with self.lock:
                    self.results.append(result)
```

```
                # Позначення завдання як виконаного
                self.task_queue.task_done()
                # Логування завершення завдання
                print("Node {}: Completed task '{}'.".format(node_id, task))
            except queue.Empty:
                # Якщо черга порожня, продовжуємо цикл
                continue

    def process_task(self, task, node_id):
        # Симуляція затримки для обробки завдання
        time.sleep(random.uniform(0.5, 1.5))
        # Обчислення MD5-хешу для завдання як симуляція результату
        task_hash = hashlib.md5(task.encode('utf-8')).hexdigest()
        # Формування результату як словника
        result = {
            "node_id": node_id,
            "task": task,
            "result_hash": task_hash,
            "timestamp": datetime.datetime.now().isoformat()
        }
        # Повернення результату обробки завдання
        return result

    def add_task(self, task):
        # Додавання нового завдання до черги
        self.task_queue.put(task)
        # Логування додавання завдання
        print("DistributedOrchestrator: Task '{}' added to queue.".format(task))

    def get_results(self):
        # Повернення копії результатів сканування
        with self.lock:
            return list(self.results)

    def shutdown(self):
        # Завершення роботи вузлів
        self.running = False

        # Очікування завершення кожного вузла з таймаутом
        for node in self.nodes:
            node.join(timeout=2)
        # Логування завершення роботи оркестратора
        print("DistributedOrchestrator: Shutdown complete.")

# ===============
# Extension 2: Cloud Sandbox Environment Manager Module
# ===============
class SandboxManager:
    # Конструктор класу SandboxManager
    def __init__(self):
        # Ініціалізація словника для зберігання активних sandbox середовищ
        self.sandboxes = {}

        # Лічильник для унікальних ідентифікаторів sandbox
        self.counter = 0

    def launch_sandbox(self, config):
        # Генерація унікального ідентифікатора для нового sandbox
        sandbox_id = "sandbox_" + str(self.counter)
        self.counter += 1
        # Створення запису про нове sandbox середовище
        sandbox = {
            "id": sandbox_id,
            "config": config,
            "status": "running",
            "start_time": datetime.datetime.now().isoformat()
        }
```

```
        # Додавання нового sandbox до словника
        self.sandboxes[sandbox_id] = sandbox
        # Логування запуску sandbox
        print("SandboxManager: Launched sandbox with ID
'{}'.".format(sandbox_id))
        # Повернення ідентифікатора нового sandbox
        return sandbox_id

    def stop_sandbox(self, sandbox_id):
        # Зупинка sandbox середовища за вказаним ідентифікатором
        if sandbox_id in self.sandboxes:
            self.sandboxes[sandbox_id]["status"] = "stopped"
            self.sandboxes[sandbox_id]["stop_time"] =
datetime.datetime.now().isoformat()
            # Логування зупинки sandbox
            print("SandboxManager: Stopped sandbox with ID
'{}'.".format(sandbox_id))
            return True
        else:
            # Логування помилки, якщо sandbox не знайдено
            print("SandboxManager: Sandbox with ID '{}' not
found.".format(sandbox_id))
            return False

    def list_sandboxes(self):
        # Виведення списку всіх sandbox середовищ
        print("SandboxManager: Listing all sandboxes.")
        for sb in self.sandboxes.values():
            print(json.dumps(sb, indent=4))

        # Повернення списку ідентифікаторів sandbox
        return list(self.sandboxes.keys())

    def get_sandbox_details(self, sandbox_id):

        # Отримання детальної інформації про конкретний sandbox
        if sandbox_id in self.sandboxes:
            details = self.sandboxes[sandbox_id]
            print("SandboxManager: Details for sandbox
'{}':".format(sandbox_id))
            print(json.dumps(details, indent=4))
            return details
        else:
            print("SandboxManager: Sandbox with ID '{}' not
found.".format(sandbox_id))
            return None

# ===============
# Extension 3: Forensic Analysis Module
# ===============
class ForensicAnalyzer:
    # Конструктор класу ForensicAnalyzer
    def __init__(self):
        # Ініціалізація списку для зберігання результатів аналізу
        self.analysis_data = []

    def analyze_file(self, file_path):
        # Аналіз файлу для виявлення потенційних загроз
        try:
            with open(file_path, "rb") as f:
                file_data = f.read()
            # Обчислення SHA256-хешу файлу
            file_hash = hashlib.sha256(file_data).hexdigest()

            # Формування результату аналізу
            analysis_result = {
```

```
                "file_path": file_path,
                "file_size": len(file_data),
                "sha256": file_hash,
                "suspicious": "no",
                "analysis_notes": "File appears to be clean upon preliminary
analysis."
            }
            # Додавання результату до списку аналізу
            self.analysis_data.append(analysis_result)
            # Логування завершення аналізу файлу
            print("ForensicAnalyzer: Analysis complete for file
'{}'.".format(file_path))
            return analysis_result
        except Exception as e:

            # Логування помилки при аналізі файлу
            print("ForensicAnalyzer: Error analyzing file '{}':
{}".format(file_path, str(e)))
            return None

    def analyze_memory_dump(self, memory_dump_data):
        # Симуляція аналізу дампу пам'яті
        analysis_result = {
            "dump_summary": "Memory dump analyzed.",
            "anomalies_detected": random.choice([True, False]),
            "details": "No significant anomalies found." if random.choice([True,
False]) else "Anomaly pattern detected."
        }
        # Додавання результату аналізу дампу до списку даних
        self.analysis_data.append(analysis_result)

        # Логування завершення аналізу дампу пам'яті
        print("ForensicAnalyzer: Memory dump analysis complete.")
        return analysis_result

    def generate_report(self):
        # Генерація детального звіту за результатами аналізу
        report = "Forensic Analysis Report\n"
        report += "=" * 30 + "\n"
        for entry in self.analysis_data:
            report += json.dumps(entry, indent=4) + "\n"
            report += "-" * 30 + "\n"

        # Логування генерації звіту
        print("ForensicAnalyzer: Report generated.")
        return report

    def archive_analysis(self, archive_path="forensic_archive.json"):
        # Архівація результатів аналізу у JSON файл
        try:
            with open(archive_path, "w") as archive_file:
                json.dump(self.analysis_data, archive_file, indent=4)
            print("ForensicAnalyzer: Analysis data archived to
'{}'.".format(archive_path))
            return True
        except Exception as e:
            print("ForensicAnalyzer: Failed to archive analysis data:
{}".format(str(e)))
            return False

# ===============
# Extension 4: Benchmarking and Performance Testing Module
# ===============
class PerformanceTester:
    # Конструктор класу PerformanceTester
    def __init__(self):
```

```
        # Ініціалізація списку результатів тестування продуктивності
        self.results = []

    def run_benchmark(self, task_function, iterations=10):
        # Запуск бенчмарку для заданої функції
        timings = []
        for i in range(iterations):
            start_time = time.time()
            # Виконання тестової функції
            task_function()
            end_time = time.time()
            elapsed = end_time - start_time
            timings.append(elapsed)
            print("PerformanceTester: Iteration {} took {:.6f}
seconds.".format(i, elapsed))
        # Обчислення середнього часу виконання
        average_time = sum(timings) / len(timings)
        result = {
            "task_function": task_function.__name__,
            "iterations": iterations,
            "average_time": average_time,
            "timings": timings
        }
        # Збереження результату тестування
        self.results.append(result)
        print("PerformanceTester: Benchmark complete for function
'{}'.".format(task_function.__name__))
        return result

    def measure_execution_time(self, func, *args, **kwargs):
        # Вимірювання часу виконання заданої функції
        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()
        execution_time = end_time - start_time
        print("PerformanceTester: Execution of '{}' took {:.6f}
seconds.".format(func.__name__, execution_time))
        return execution_time, result

    def compare_performance(self, functions_list):
        # Порівняння продуктивності декількох функцій
        comparison_results = {}
        for func in functions_list:
            exec_time, _ = self.measure_execution_time(func)
            comparison_results[func.__name__] = exec_time
            print("PerformanceTester: Function '{}' execution time: {:.6f}
seconds.".format(func.__name__, exec_time))
        print("PerformanceTester: Performance comparison complete.")
        return comparison_results

    def generate_performance_report(self):
        # Генерація звіту за результатами тестування продуктивності
        report = "Performance Testing Report\n"
        report += "=" * 30 + "\n"
        for result in self.results:
            report += json.dumps(result, indent=4) + "\n"
            report += "-" * 30 + "\n"
        print("PerformanceTester: Performance report generated.")
        return report

# ===============
# Extension 5: Synthetic Data Generator Module
# ===============
class SyntheticDataGenerator:
    # Конструктор класу SyntheticDataGenerator
    def __init__(self):
        # Ініціалізація параметрів для генерації даних
```

```
        self.numeric_range = (0, 1000)
        self.text_characters = string.ascii_letters + string.digits
        self.time_format = "%Y-%m-%d %H:%M:%S"

    def generate_numeric_data(self, size=100):
        # Генерація списку числових даних
        numeric_data = []
        for i in range(size):
            number = random.randint(self.numeric_range[0],
self.numeric_range[1])
            numeric_data.append(number)
            print("SyntheticDataGenerator: Generated number {}: {}".format(i,
number))
        return numeric_data

    def generate_text_data(self, size=50, length=10):
        # Генерація списку текстових даних
        text_data = []
        for i in range(size):
            text = ''.join(random.choice(self.text_characters) for _ in
range(length))
            text_data.append(text)
            print("SyntheticDataGenerator: Generated text {}: {}".format(i,
text))
        return text_data

    def generate_mixed_data(self, size=100):
        # Генерація змішаних даних (числові та текстові)
        mixed_data = []
        for i in range(size):
            if i % 2 == 0:
                value = random.randint(self.numeric_range[0],
self.numeric_range[1])
            else:
                value = ''.join(random.choice(self.text_characters) for _ in
range(8))
            mixed_data.append(value)
            print("SyntheticDataGenerator: Generated mixed data {}:
{}".format(i, value))
        return mixed_data

    def generate_time_series_data(self, size=24):
        # Генерація даних часових рядів
        time_series = []
        current_time = datetime.datetime.now()
        for i in range(size):
            timestamp = (current_time +
datetime.timedelta(hours=i)).strftime(self.time_format)
            value = random.uniform(0, 100)
            entry = {"timestamp": timestamp, "value": value}
            time_series.append(entry)
            print("SyntheticDataGenerator: Generated time series entry {}:
{}".format(i, entry))
        return time_series
    def generate_complex_structure_data(self, size=10):
        # Генерація складних структурованих даних
        complex_data = []
        for i in range(size):
            record = {
                "id": i,
                "metrics": {
                    "cpu": random.uniform(0, 100),
                    "memory": random.uniform(0, 100),
                    "disk": random.uniform(0, 100)
                },
```

```
      "logs": [self.generate_text_data(size=3, length=15) for _ in range(2)],
      "timestamp": datetime.datetime.now().isoformat()
            }
            complex_data.append(record)
            print("SyntheticDataGenerator: Generated complex structure record
{}: {}".format(i, record))
        return complex_data
# ===============
# Main function to test all 5 extensions
# ===============
def main():
    # Тестування Distributed Orchestrator
    print("\n=== Testing Distributed Orchestrator ===")
    orchestrator = DistributedOrchestrator()
    tasks = ["Scan File A", "Scan File B", "Scan File C", "Scan File D", "Scan
File E"]
    for task in tasks:
        orchestrator.add_task(task)
    # Очікування обробки завдань
    time.sleep(5)
    results = orchestrator.get_results()
    print("Distributed Orchestrator Results:")

    for res in results:
        print(json.dumps(res, indent=4))
    orchestrator.shutdown()
    # Тестування Sandbox Manager
    print("\n=== Testing Sandbox Manager ===")
    sandbox_manager = SandboxManager()
    config1 = {"os": "Ubuntu 20.04", "resources": {"cpu": 2, "memory": "2GB"}}
    config2 = {"os": "Windows Server 2019", "resources": {"cpu": 4, "memory":
"4GB"}}
    sandbox_id1 = sandbox_manager.launch_sandbox(config1)
    sandbox_id2 = sandbox_manager.launch_sandbox(config2)
    sandbox_manager.list_sandboxes()
    sandbox_manager.get_sandbox_details(sandbox_id1)
    sandbox_manager.stop_sandbox(sandbox_id1)
    sandbox_manager.get_sandbox_details(sandbox_id1)
    # Тестування Forensic Analyzer
    print("\n=== Testing Forensic Analyzer ===")
    forensic_analyzer = ForensicAnalyzer()
    # Створення тимчасового файлу для аналізу
    temp_file = "temp_test_file.txt"
    with open(temp_file, "w", encoding="utf-8") as f:
        f.write("Це тестовий файл для Forensic Analysis Module. Інформація для
аналізу.")
    forensic_result = forensic_analyzer.analyze_file(temp_file)
    forensic_analyzer.analyze_memory_dump(" memory dump data")
    report = forensic_analyzer.generate_report()
    print(report)
    forensic_analyzer.archive_analysis("forensic_archive_test.json")
    # Видалення тимчасового файлу
    try:
        import os
        os.remove(temp_file)
    except Exception as e:
        print("ForensicAnalyzer: Error removing temporary file:", e)
    # Тестування Performance Tester
    print("\n=== Testing Performance Tester ===")
    performance_tester = PerformanceTester()
    # Визначення тестової функції для бенчмарку
    def sample_task():
        total = 0
        for i in range(10000):
            total += math.sqrt(i)
        return total
```

```
    benchmark_result = performance_tester.run_benchmark(sample_task,
iterations=5)
    comparison = performance_tester.compare_performance([sample_task, lambda:
sample_task()])
    performance_report = performance_tester.generate_performance_report()
    print(performance_report)

    # Тестування Synthetic Data Generator
    print("\n=== Testing Synthetic Data Generator ===")
    data_generator = SyntheticDataGenerator()
    numeric_data = data_generator.generate_numeric_data(size=10)
    text_data = data_generator.generate_text_data(size=5, length=12)
    mixed_data = data_generator.generate_mixed_data(size=8)
    time_series = data_generator.generate_time_series_data(size=6)
    complex_data = data_generator.generate_complex_structure_data(size=3)

   # Виведення згенерованих даних
    print("SyntheticDataGenerator: Numeric Data:", numeric_data)
    print("SyntheticDataGenerator: Text Data:", text_data)
    print("SyntheticDataGenerator: Mixed Data:", mixed_data)
    print("SyntheticDataGenerator: Time Series Data:", time_series)
    print("SyntheticDataGenerator: Complex Structure Data:", complex_data)

# ===============
# Запуск основної функції при виконанні скрипта
# ===============
if __name__ == "__main__":
    main()
```

Файл Ext2.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Extended Modules Implementation: Additional 5 Extensions 2
# The extensions implemented are:
# 1. Machine Learning Detection Module
# 2. User Management and Authentication Module
# 3. Intrusion Detection System Module
# 4. Report Generation Module
# 5. Dashboard and Analytics Module
# ================================================================
import time
import random
import hashlib
import json
import threading
import datetime
import queue
import math
import string

# ===============
# Module 1: Machine Learning Detection Module
# ===============
class MachineLearningDetector:
    # Constructor for MachineLearningDetector
    def __init__(self):
        # Initialize model, training data, and logs
        self.model = None
        self.training_data = []
        self.training_labels = []
        self.is_trained = False
        self.training_logs = []
        self.threshold = 0.5
        print("MachineLearningDetector: Initialized with default parameters.")

    def generate_synthetic_training_data(self, num_samples=100):
        # Generate synthetic training data samples
        print("MachineLearningDetector: Generating synthetic training data.")
        for i in range(num_samples):
            # Each sample is a list of 10 random features
            features = [random.random() for _ in range(10)]
            # Randomly assign label 0 (clean) or 1 (virus)
            label = random.choice([0, 1])
            self.training_data.append(features)
            self.training_labels.append(label)
            self.training_logs.append("Sample {}: Features = {}, Label =
{}".format(i, features, label))
        print("MachineLearningDetector: Generated {} training
samples.".format(num_samples))

    def train_model(self):
        # Train a  model using synthetic data
        print("MachineLearningDetector: Starting model training.")
        if not self.training_data:
            self.generate_synthetic_training_data()
        # training by generating a  weight vector
        weight_vector = [random.random() for _ in range(10)]
        self.model = weight_vector
        self.is_trained = True
        self.training_logs.append("Model trained with weight vector:
{}".format(weight_vector))
        print("MachineLearningDetector: Model training complete. Model is now
trained.")
```

```
    def predict(self, features):
        # Predict the outcome for given features using the  model
        if not self.is_trained:
            print("MachineLearningDetector: Model not trained. Training now.")
            self.train_model()
        # Compute a  dot product as score
        score = sum(f * w for f, w in zip(features, self.model))
        normalized_score = score / (sum(self.model) + 1e-6)
        print("MachineLearningDetector: Computed score = {:.6f}, normalized
score = {:.6f}".format(score, normalized_score))
        prediction = 1 if normalized_score > self.threshold else 0
        print("MachineLearningDetector: Prediction = {}".format("Virus Detected"
if prediction == 1 else "Clean"))
        return prediction

    def save_model(self, file_path="ml_model.json"):
        # Save the model and training logs to a JSON file
        if not self.is_trained:
            print("MachineLearningDetector: Cannot save model as it is not
trained.")
            return False
        data_to_save = {
            "model": self.model,
            "training_logs": self.training_logs,
            "threshold": self.threshold
        }
        try:
            with open(file_path, "w") as f:
                json.dump(data_to_save, f, indent=4)
             print("MachineLearningDetector: Model saved successfully to
'{}'.".format(file_path))
            return True
        except Exception as e:
            print("MachineLearningDetector: Error saving model: {}".format(e))
            return False

    def load_model(self, file_path="ml_model.json"):
        # Load the model and training logs from a JSON file
        try:
            with open(file_path, "r") as f:
                data_loaded = json.load(f)
            self.model = data_loaded.get("model", None)
            self.training_logs = data_loaded.get("training_logs", [])
            self.threshold = data_loaded.get("threshold", 0.5)
            self.is_trained = self.model is not None
            print("MachineLearningDetector: Model loaded successfully from
'{}'.".format(file_path))
            return True
        except Exception as e:
            print("MachineLearningDetector: Error loading model: {}".format(e))
            return False

# ===============
# Module 2: User Management and Authentication Module
# ===============
class UserManager:
    # Constructor for UserManager
    def __init__(self):
        # Dictionary to store user details and sessions
        self.users = {}
        self.sessions = {}
        self.user_id_counter = 1
        self.activity_log = []
        print("UserManager: Initialized user management system.")
```

```
    def hash_password(self, password, salt=None):
        # Hash the password with an optional salt for security
        if salt is None:
            salt = ''.join(random.choices(string.ascii_letters + string.digits,
k=8))
        salted_password = password + salt
        hashed = hashlib.md5(salted_password.encode('utf-8')).hexdigest()
        print("UserManager: Hashed password with salt '{}' resulting in hash
'{}'.".format(salt, hashed))
        return hashed, salt

    def register_user(self, username, password, email):
        # Register a new user if username does not exist
        if username in self.users:
            print("UserManager: Registration failed. Username '{}' already
exists.".format(username))
            return False
        hashed_password, salt = self.hash_password(password)
        user_details = {
            "user_id": self.user_id_counter,
            "username": username,
            "password_hash": hashed_password,
            "salt": salt,
            "email": email,
            "registered_at": datetime.datetime.now().isoformat()
        }
        self.users[username] = user_details
        self.user_id_counter += 1
        self.activity_log.append("Registered user '{}'.".format(username))
        print("UserManager: User '{}' registered
successfully.".format(username))
        return True

    def login_user(self, username, password):
        # Log in a user if credentials match
        user = self.users.get(username, None)
        if not user:
            print("UserManager: Login failed. Username '{}' not
found.".format(username))
            return None
        hashed_password, _ = self.hash_password(password, user["salt"])
        if hashed_password == user["password_hash"]:
            token = ''.join(random.choices(string.ascii_letters + string.digits,
k=16))
            self.sessions[token] = username
            self.activity_log.append("User '{}' logged in with token
'{}'.".format(username, token))
            print("UserManager: User '{}' logged in successfully. Session token:
'{}'.".format(username, token))
            return token
        else:
            print("UserManager: Login failed. Incorrect password for username
'{}'.".format(username))
            return None

    def logout_user(self, token):
        # Log out the user by removing the session token
        if token in self.sessions:
            username = self.sessions.pop(token)
            self.activity_log.append("User '{}' logged out.".format(username))
            print("UserManager: User '{}' logged out
successfully.".format(username))
            return True
        else:
            print("UserManager: Logout failed. Session token '{}' not
found.".format(token))
            return False
```

```
    def list_users(self):
        # List all registered users
        print("UserManager: Listing all registered users:")
        for username, details in self.users.items():
            print(json.dumps(details, indent=4))
        return list(self.users.keys())

# ===============
# Module 3: Intrusion Detection System Module
# ===============
class IntrusionDetectionSystem:
    # Constructor for IntrusionDetectionSystem
    def __init__(self):
        # Queue for incoming network packets and alerts list
        self.packet_queue = queue.Queue()
        self.alerts = []
        self.monitoring = False
        self.monitor_thread = None
        print("IntrusionDetectionSystem: Initialized IDS.")

    def start_monitoring(self):
        # Start monitoring network traffic in a separate thread
        self.monitoring = True
        self.monitor_thread = threading.Thread(target=self.monitor_network)
        self.monitor_thread.daemon = True
        self.monitor_thread.start()
        print("IntrusionDetectionSystem: Network monitoring started.")

    def stop_monitoring(self):
        # Stop network monitoring
        self.monitoring = False
        if self.monitor_thread:
            self.monitor_thread.join(timeout=2)
        print("IntrusionDetectionSystem: Network monitoring stopped.")

    def _packet_injection(self, packet_data):
        # Simulate the injection of a network packet
        self.packet_queue.put(packet_data)
        print("IntrusionDetectionSystem: Packet injected:
{}".format(packet_data))

    def monitor_network(self):
        # Monitor network traffic by processing packets from the queue
        print("IntrusionDetectionSystem: Monitoring network traffic...")
        while self.monitoring:
            try:
                packet = self.packet_queue.get(timeout=1)
                self.analyze_packet(packet)
                self.packet_queue.task_done()
            except queue.Empty:
                continue

    def analyze_packet(self, packet):
        # Analyze a network packet for suspicious content
        print("IntrusionDetectionSystem: Analyzing packet: {}".format(packet))
        if "attack" in packet.lower():
            alert_msg = "Suspicious packet detected: {}".format(packet)
            self.alerts.append(alert_msg)
            self.send_alert(alert_msg)
        else:
            print("IntrusionDetectionSystem: Packet appears normal.")

    def send_alert(self, alert_message):
        # Simulate sending an alert for suspicious packet
        timestamp = datetime.datetime.now().isoformat()
        full_alert = "[{}] ALERT: {}".format(timestamp, alert_message)
        self.alerts.append(full_alert)
        print("IntrusionDetectionSystem: Alert sent: {}".format(full_alert))
```

```
    def get_alerts(self):
        # Retrieve all generated alerts
        print("IntrusionDetectionSystem: Retrieving all alerts.")
        return list(self.alerts)

# ===============
# Module 4: Report Generation Module
# ===============
class ReportGenerator:
    # Constructor for ReportGenerator
    def __init__(self):
        # Initialize report sections and metadata
        self.sections = []
        self.title = "System Report"
        self.generated_at = datetime.datetime.now().isoformat()
        self.notes = "This report is generated automatically."
        print("ReportGenerator: Initialized report generator.")

    def add_section(self, heading, content):
        # Add a new section to the report
        section = {
            "heading": heading,
            "content": content,
            "timestamp": datetime.datetime.now().isoformat()
        }
        self.sections.append(section)
        print("ReportGenerator: Added section '{}'.".format(heading))

    def generate_text_report(self):
        # Generate a text-based report from the sections
        report = "Report Title: {}\n".format(self.title)
        report += "Generated At: {}\n".format(self.generated_at)
        report += "Notes: {}\n".format(self.notes)
        report += "=" * 50 + "\n"
        for sec in self.sections:
            report += "Section: {}\n".format(sec["heading"])
            report += "Time: {}\n".format(sec["timestamp"])
            report += sec["content"] + "\n"
            report += "-" * 50 + "\n"
        print("ReportGenerator: Text report generated.")
        return report

    def generate_json_report(self):
        # Generate a JSON formatted report
        report_data = {
            "title": self.title,
            "generated_at": self.generated_at,
            "notes": self.notes,
            "sections": self.sections
        }
        json_report = json.dumps(report_data, indent=4)
        print("ReportGenerator: JSON report generated.")
        return json_report

    def save_report(self, file_path="system_report.txt", format_type="text"):
        # Save the report to a file in the chosen format
        try:
            if format_type == "text":
                report_content = self.generate_text_report()
            elif format_type == "json":
                report_content = self.generate_json_report()
            else:
                report_content = self.generate_text_report()
            with open(file_path, "w") as f:
                f.write(report_content)
            print("ReportGenerator: Report saved to '{}'.".format(file_path))
            return True
```

```
        except Exception as e:
            print("ReportGenerator: Error saving report: {}".format(e))
            return False

# ===============
# Module 5: Dashboard and Analytics Module
# ===============
class DashboardAnalytics:
    # Constructor for DashboardAnalytics
    def __init__(self):
        # Initialize metrics storage and dashboard log
        self.metrics = {}
        self.dashboard_log = []
        self.timestamps = []
        self.initialize_dashboard()
        print("DashboardAnalytics: Dashboard initialized.")

    def initialize_dashboard(self):
        # Set up default metrics and log initialization
        self.metrics = {
            "cpu_usage": [],
            "memory_usage": [],
            "disk_io": [],
            "network_traffic": []
        }
        self.timestamps = []
        self.dashboard_log.append("Dashboard initialized at
{}".format(datetime.datetime.now().isoformat()))

    def collect_metric(self):
        # Simulate metric collection
        cpu = random.uniform(0, 100)
        memory = random.uniform(0, 100)
        disk = random.uniform(0, 100)
        network = random.uniform(0, 1000)
        timestamp = datetime.datetime.now().isoformat()
        self.metrics["cpu_usage"].append(cpu)
        self.metrics["memory_usage"].append(memory)
        self.metrics["disk_io"].append(disk)
        self.metrics["network_traffic"].append(network)
        self.timestamps.append(timestamp)
        log_entry = "Collected metrics at {}: CPU={:.2f}%, Memory={:.2f}%,
Disk={:.2f}%, Network={:.2f}KB/s".format(
            timestamp, cpu, memory, disk, network)
        self.dashboard_log.append(log_entry)
        print("DashboardAnalytics: " + log_entry)

    def analyze_metrics(self):
        # Perform basic analysis on collected metrics
        analysis = {}
        analysis["avg_cpu"] = sum(self.metrics["cpu_usage"]) /
len(self.metrics["cpu_usage"]) if self.metrics["cpu_usage"] else 0
        analysis["avg_memory"] = sum(self.metrics["memory_usage"]) /
len(self.metrics["memory_usage"]) if self.metrics["memory_usage"] else 0
        analysis["avg_disk_io"] = sum(self.metrics["disk_io"]) /
len(self.metrics["disk_io"]) if self.metrics["disk_io"] else 0
        analysis["avg_network_traffic"] = sum(self.metrics["network_traffic"]) /
len(self.metrics["network_traffic"]) if self.metrics["network_traffic"] else 0
        self.dashboard_log.append("Metrics analyzed: " + json.dumps(analysis))
        print("DashboardAnalytics: Metrics analysis complete:
{}".format(analysis))
        return analysis

    def display_dashboard(self):
        # Display the dashboard with collected metrics and analysis
        print("\n=== Dashboard Analytics ===")
        print("Timestamps: ", self.timestamps)
```

```
        print("Metrics: ", json.dumps(self.metrics, indent=4))
        analysis = self.analyze_metrics()
        print("Analysis: ", json.dumps(analysis, indent=4))
        print("============================")

    def generate_dashboard_report(self):
        # Generate a detailed dashboard report as text
        report = "Dashboard Report\n"
        report += "Generated At:
{}\n".format(datetime.datetime.now().isoformat())
        report += "=" * 50 + "\n"
        report += "Collected Timestamps:\n" + "\n".join(self.timestamps) + "\n"
        report += "\nCollected Metrics:\n" + json.dumps(self.metrics, indent=4)
+ "\n"
        analysis = self.analyze_metrics()
        report += "\nAnalysis:\n" + json.dumps(analysis, indent=4) + "\n"
        report += "\nDashboard Log:\n" + "\n".join(self.dashboard_log) + "\n"
        report += "=" * 50 + "\n"
        print("DashboardAnalytics: Dashboard report generated.")
        return report

    def save_dashboard_report(self, file_path="dashboard_report.txt"):
        # Save the dashboard report to a file
        report_content = self.generate_dashboard_report()
        try:
            with open(file_path, "w") as f:
                f.write(report_content)
            print("DashboardAnalytics: Dashboard report saved to
'{}'.".format(file_path))
            return True
        except Exception as e:
            print("DashboardAnalytics: Error saving dashboard report:
{}".format(e))
            return False

# ===============
# Main function to test the 5 additional modules
# ===============
def main():
    print("\n=== Testing Machine Learning Detection Module ===")
    ml_detector = MachineLearningDetector()
    ml_detector.generate_synthetic_training_data(50)
    ml_detector.train_model()
    sample_features = [random.random() for _ in range(10)]
    prediction = ml_detector.predict(sample_features)
    ml_detector.save_model("ml_model_test.json")
    ml_detector.load_model("ml_model_test.json")

    print("\n=== Testing User Management and Authentication Module ===")
    user_manager = UserManager()
    user_manager.register_user("alice", "password123", "alice@example.com")
    user_manager.register_user("bob", "securePass", "bob@example.com")
    token_alice = user_manager.login_user("alice", "password123")
    token_bob = user_manager.login_user("bob", "wrongPass")
    user_manager.list_users()
    user_manager.logout_user(token_alice)

    print("\n=== Testing Intrusion Detection System Module ===")
    ids = IntrusionDetectionSystem()
    ids.start_monitoring()
    ids.simulate_packet_injection("Normal packet data")
    ids.simulate_packet_injection("This packet contains an attack signature")
    time.sleep(3)
    alerts = ids.get_alerts()
    print("Intrusion Detection Alerts:")
    for alert in alerts:
        print(alert)
    ids.stop_monitoring()
```

```
    print("\n=== Testing Report Generation Module ===")
    report_gen = ReportGenerator()
    report_gen.add_section("System Overview", "The system is operational with
all modules running as expected.")
    report_gen.add_section("Security Events", "Multiple security events have
been logged in the system.")
    text_report = report_gen.generate_text_report()
    json_report = report_gen.generate_json_report()
    print("Text Report:\n", text_report)
    print("JSON Report:\n", json_report)
    report_gen.save_report("system_report_test.txt", format_type="text")

    print("\n=== Testing Dashboard and Analytics Module ===")
    dashboard = DashboardAnalytics()
    for i in range(5):
        dashboard.collect_metric()
        time.sleep(1)
    dashboard.display_dashboard()
    dashboard.save_dashboard_report("dashboard_report_test.txt")

# ===============
# Entry point for the script
# ===============
if __name__ == "__main__":
    main()
```