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 кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA”

Виконав здобувач вищої освіти
IV курсу, групи КБ-21
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Юрченко Д.І.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Буравченко К.О.
«____» _____ 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 *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

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

Юрченку Данилу Ігоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA*

2. Керівник роботи *Буравченко Костянтин Олегович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 57-02 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 кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA*

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 кібербезпеки в промислову експлуатацію.

6. Висновки

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

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*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
_____ Буравченко К.О.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Юрченко Д.І.
(прізвище та ініціали)

АНОТАЦІЯ

Юрченко Д.І. Програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

Метою розробки є програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

Результат роботи – програмна реалізація системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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: кібербезпека, VSA

ABSTRACT

Yurchenko D.I. Software for the cybersecurity system of information security gateways using VSA technology. 125 Cybersecurity. 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, which is intended for the cybersecurity system of information security gateways using VSA technology.

The purpose of the development is the software for the cybersecurity system of information security gateways using VSA technology.

The result of the work is the software implementation of the cybersecurity system of information security gateways using VSA technology.

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 OS Windows 10/11.

The program was developed in the Python environment.

Keywords: cybersecurity, VSA

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ГПВЧ – генератор псевдовипадкових чисел

ДСТ – державний стандарт

ЕОМ – електронна обчислювальна машина

ІБ – інформаційна безпека

ІС – інформаційна система

КСЗІ – комплекс системи захисту інформації

НЗД – найбільший загальний дільник

НСД – несанкціонований доступ

ПЗ – програмне забезпечення

ПЗП – Постійно Запам'ятовувальний Пристрій

EEPROM – Electronically EPROM

EPROM – Erasable Programmable ROM або як Electrically Programmable ROM

Flash – Flash Erase EEPROM

NVRWM – nonvolatile read-write memory

PROM – Programmable ROM

RAM – Random Access Memory

ROM – Read Only Memory

ВСТУП

Актуальність теми. Security Appliance (SA) – це просто «пристрій», призначений для захисту комп'ютерних мереж від небажаного трафіку. Цей пристрій може бути активним і блокувати небажаний трафік. Це стосується, наприклад, брандмауерів і фільтрів вмісту. Пристрій безпеки також може бути пасивним. Тут його роль – просто виявлення та звітування. Гарним прикладом є системи виявлення вторгнень. Якщо SA відповідає за сканування мережі та виявлення потенційних порушень (наприклад, тестування на проникнення), SA можна кваліфікувати як профілактичний. У наш час пристрої безпеки поєднують різні функції безпеки, включаючи брандмауер, фільтрацію вмісту та виявлення вторгнень. З цієї причини вони більш відомі як системи Уніфікованого управління загрозами (UTM). Звичайні пристрої безпеки (зокрема брандмауери) були розгорнуті на межі мережі, щоб перевірити трафік, що спрямовується до цієї мережі. Оскільки мережі стають складнішими, часто доводиться розміщувати ці пристрої між кількома сегментами мережі. З NFV і хмарними обчисленнями ситуація стає складнішою, оскільки цілі мережі або сегменти мережі можуть бути повністю розміщені у віртуальному середовищі. Як наслідок, пристрої безпеки також повинні захищати віртуальні середовища на додаток до фізичних мереж. Тягар цього завдання можуть виконувати пристрої безпеки, що працюють на віртуальних машинах. Віртуальний пристрій безпеки (vSA) – це служба безпеки мережі, яка повністю працює в віртуалізованому середовищі.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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

– Огляд існуючих систем шлюзів інформаційної безпеки з використанням технології VSA.

– Дослідження системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

– Програмна реалізація системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі шлюзів інформаційної безпеки з використанням технології VSA.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA, 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

Оскільки віртуальний пристрій безпеки може охоплювати різні технології безпеки, тип безпеки, який використовується в цьому контексті, є більш важливим, коли йдеться про продуктивність. У минулому продуктивність приладу досягалася за допомогою спеціального обладнання. У віртуалізованих середовищах це неможливо з тієї причини, що різні додатки можуть працювати в одній операційній системі та конкурувати за однакові апаратно-обчислювальні ресурси.

У даній роботі ми обговорюємо початкові кроки до віртуалізації пристроїв безпеки. Дійсно, необхідно вирішити різні проблеми, пов'язані з розгортанням таких пристроїв.

Складність пристрою безпеки: визначити, чи складається пристрій з одного чи кількох компонентів.

Внутрішній зв'язок: якщо SA включає різні компоненти, як забезпечити належний зв'язок між віртуальними машинами, на яких запущено різні компоненти.

Інтеграція з віртуальною інфраструктурою: під час налаштування компонентів SA нам також потрібно переконатися, що віртуальні платформи, такі як OpenStack, сприяють необхідній інтеграції, включаючи IP. надання адрес, дзеркальне відображення трафіку, якщо необхідно, тощо.

Вплив на продуктивність: тут може знадобитися вибір технологій безпеки, які можна віртуалізувати. Наприклад, брандмауери можуть бути «хорошим» кандидатом, оскільки вони перевіряють невеликі обсяги даних (заголовки пакетів) і, як правило, не мають статусу.

Крім того, вони можуть працювати як частина операційної системи, яка не

впливатиме на продуктивність системи. Щоб мати реалістичний сценарій і мати справу з ширшим діапазоном атак, ми пропонуємо віртуальний пристрій безпеки, що складається з віртуального брандмауера, який використовується як точка входу в мережу та відповідатиме за «просту» фільтрацію пакетів. Система виявлення вторгнень, яка обробляє більш глибоку перевірку пакетів і видає сповіщення у разі зловмисного трафіку.

1.2 Область застосування

Пристрої безпеки повинні бути розміщені в мережевому трафіку. Це стосується як пасивних, так і активних мережевих елементів. В ідеалі весь трафік має проходити через елементи безпеки, тому ці елементи зазвичай розміщуються поблизу граничного маршрутизатора клієнта (CE). Це стосується як житлового, так і корпоративного сценаріїв, хоча в першому випадку вимоги набагато менші, тому пристрої безпеки зазвичай інтегруються в CE.

У корпоративному сценарії вони складаються зі спеціального спеціального обладнання та розмішуються в домені користувача.

Процес віртуалізації пристроїв безпеки сам по собі не є вимогою до обслуговування. Тому він має бути максимально прозорим для мережі, зберігаючи застаріле розміщення та функціональність. З парадигмою NFV очікуються деякі зміни в інфраструктурі, такі як поява NFVI-PoPs. Це невеликі розподілені центри обробки даних, які можна використовувати для розгортання VNF. Фактична кількість і розміщення точок NFVI-PoP все ще активно обговорюється, але зазвичай вони повинні розташовуватися поблизу межових маршрутизаторів провайдера (PE). Зважаючи на це, розміщення віртуальних аналогів пристроїв безпеки (vSA) може призвести до двох різних випадків: невіртуалізований CE – у цьому випадку vSA буде розміщено між CE та PE. Ця ситуація змушує змінити традиційну топологію мережі. Тим не менш, ця зміна має бути майже прозорою, а функціональність vSA має бути збережена.

Віртуалізований CE – у цьому випадку логічне розміщення vSA та CE ідентично традиційному сценарію, і обидві функції переміщено до NFVI-PoP. Крім того, попередній фізичний CE потрібно замінити мостом L2, щоб розширити широкомовний домен клієнтської мережі до віртуального середовища, розташованого в NFVI-PoP представляє інші проблеми, які виходять за межі цієї роботи. Крім того, з точки зору vSA, використання чи ні віртуального CE має бути прозорим.

Нарешті, розглядатиметься лише випадок використання на підприємстві, оскільки сценарій vSA у житлових приміщеннях буде перебільшеним. У цьому випадку використання компанія хоче скористатися перевагами віртуалізації для розгортання функцій безпеки мережі без шкоди для продуктивності та функціональності. Крім того, переходячи до хмарного середовища, компанія бажає підтримувати застарілу мережеву інфраструктуру якомога ближче до реальності. Ця мережева інфраструктура може включати ресурси TelcoVPN, на які нові послуги не повинні впливати

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA, 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

Проведемо огляд шлюзів безпеки. Шлюзи безпеки Check Point є потужною й закінченою платформою для впровадження й керування архітектурою Software Blade, що покриває практично всі аспекти інформаційної безпеки бізнесу будь-якого розміру. Всі шлюзи безпеки, завдяки підтримці уніфікованій архітектурі Software Blades, дозволяють захистити організацію від швидко мінливих погроз і управляти всім різноманіттям рішень із єдиної консолі керування.

2200 Appliance

Шлюзи безпеки Check Point 2200 Appliance надає захист на рівні підприємства для невеликих компаній, при цьому маючи лідируюче співвідношення ціна/продуктивність у форм-факторі компактного настільного пристрою. Ідеальний вибір для захисту невеликих офісів або філій великої компанії.

4000 Appliances

Сучасний шлюз безпеки підприємства повинен бути більш, ніж просто міжмережний екран – він повинен використовувати різні технології для захисту мережної інфраструктури від нових, сучасних погроз. Лінійка шлюзів Check Point 4000 Appliances завдяки наявності слота розширення (додаткові мережні інтерфейси) і підтримці багатоядерності процесора має кращу продуктивність у своєму класі.

12000 Appliances

Ці шлюзи рівня ЦОД з підтримкою багатоядерності, технологією прискорення обробки трафіку, наявністю надлишкових (відказостійких)

компонентів і підвищеною продуктивністю спеціально розроблені для високонавантажених мереж, що вимагають підвищеної надійності.

21400 Appliance

Шлюз 21400 Appliance пропонує неперевершену масштабованість, надійність і працездатність при високій продуктивності й великій кількості мережних інтерфейсів. Ідеальний для захисту великих мереж і ЦОД.

61000 Security System

Check Point 61000 Security System представляє найшвидший в індустрії шлюз безпеки. Це пристрій операторського класу пропонує масштабовану продуктивність для ЦОД, ринку телекомунікації й провайдерів хмарних сервісів.

Програмні шлюзи безпеки (Software Security Gateways)

Check Point пропонує як апаратні рішення (Appliance), так і можливість установити програмний шлюз із необхідною ліцензією на вже існуючий у замовника x86 сервер. Нижче наведені типові шлюзи (контейнер + набір блейдів), однак Ви абсолютно вільні у виборі як контейнера, так і блейдів.

Ці шлюзи повинні управлятися з окремого сервера SmartCenter, якщо ж такого в компанії поки немає (Check Point купується вперше), те, можливо, доцільно купити зв'язування (bundle) зі шлюзу й сервера керування – зв'язування коштує дешевше, ніж 2 компоненти окремо.

Series 100 – ідеальний вибір для маленького офісу. Ліцензія на 1 ядро, до 50 що захищаються ПК усередині мережі й рекомендовано для використання до 8-мі портів на сервері.

Series 200 – раціональний вибір для компаній середнього розміру. Підтримка 2-х ядер, 500 співробітників; рекомендована кількість портів – до 12 шт.

Series 400 – розроблено для високопродуктивних мереж, таких як мережі кампусів і ЦОД. Оптимізовано для 8-мі ядер.

Series 800 – розроблено для особливо високопродуктивних мереж, таких як мережі великих кампусів і ЦОД. Оптимізовано для 8-мі ядер.

Series 1200 – розроблено для особливо високопродуктивних мереж, таких як мережі великих кампусів і ЦОД. Оптимізовано для 12-ти ядер.

SG103

Firewall, VPN, IPS, Application Control, Identity Awareness. Міжмережний екран початкового рівня для захисту маленького офісу або філії.

SG108

Firewall, VPN, IPS, Application Control, Identity Awareness , Anti-Spam & Email Security, URL Filtering, Antivirus & Anti-Malware. Оптимальний шлюз безпеки, що надає захист рівня Total Security для маленького офісу або філії.

SG205i

Firewall, VPN, IPS, Application Control, Identity Awareness. Міжмережний екран початкового рівня для захисту компаній середнього розміру.

SG205U

Firewall, VPN, IPS, Application Control, Identity Awareness. Міжмережний екран початкового рівня для захисту компаній середнього розміру з кількістю співробітників більше 500 чоловік.

SG207i

Firewall, IPSEC VPN, IPS, Application Control, Identity Awareness, Advanced Networking, Acceleration & Clustering. Продуктивний шлюз безпеки для компаній середнього розміру й розвитий мережною інфраструктурою.

SG209

Firewall, VPN, IPS, Application Control, Identity Awareness, Anti-Spam & Email Security, URL Filtering, Antivirus & Anti-Malware, Acceleration & Clustering. Оптимальний шлюз безпеки, що надає захист рівня Total Security для компаній середнього розміру

SG407i

Firewall, VPN, IPS, Application Control, Identity Awareness, Advanced Networking, Acceleration & Clustering. Високопродуктивний шлюз безпеки для компаній будь-якого розміру.

SG409

Firewall, VPN, IPS, Application Control, Identity Awareness, Anti-Spam & Email Security, URL Filtering, Antivirus & Anti-Malware, Acceleration & Clustering. Оптимальний шлюз безпеки, що надає захист рівня Total Security для компаній будь-якого розміру й потребуючої підвищеної надійності.

SG807

Firewall, VPN, IPS, Application Control, Identity Awareness, Advanced Networking, Acceleration & Clustering. Високопродуктивний шлюз безпеки для високонавантажених сегментів.

SG1207

Firewall, VPN, IPS, Application Control, Identity Awareness, Advanced Networking, Acceleration & Clustering. Найбільш високопродуктивний шлюз безпеки для найбільше високонавантажених середовищ.

Trout

Trout Cyberbox – це інноваційний операційний брандмауер наступного покоління, призначений для підвищення безпеки існуючих сайтів без потреби в значній зміні проводки. Це робить його економічно ефективним і зручним варіантом для оновлення існуючих сайтів.

Однією з його видатних особливостей є можливість розгортання безпечних анклавів навколо активів – від підключених машин до камер, дверей, систем сигналізації… – забезпечуючи додатковий рівень вдосконаленої безпеки. Ці анклави допомагають захистити чутливі зони від потенційних загроз і несанкціонованого доступу.

На додаток до розширених функцій безпеки Trout Cyberbox містить посібники з відповідями. Ці посібники дозволяють операторам ефективно вирішувати інциденти безпеки. Маючи ці посібники, організації можуть швидко й ефективно реагувати на будь-які порушення безпеки або аварії, мінімізуючи потенційну шкоду та простої.

Гарна безпека мережі схожа на Kouign-amann : проста, багатошарова, стійка. Почніть із простого брандмауера, який обробляє найнеобхідніше, а потім вдосконаліть його за допомогою додаткового мережевого рішення (наприклад, Trout), щоб увімкнути внутрішню сегментацію, покращити видимість і продуктивність.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Python – динамічна інтерпретована об'єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії

мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп'ютері. Одна з основних функцій процесора – це обробка даних згідно комп'ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп'ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор. Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

– Mu – редактор коду Python для програмістів-початківців. Для користувачів Linux, Windows і macOS.

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 кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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 кібербезпеки 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 кібербезпеки 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

В обчислювальній техніці віртуальна машина – це емуляція комп'ютерної системи. Програмне забезпечення, що містить віртуальне середовище або гіпервізор, який віртуальна машина запускає на абстрактних фізичних ресурсах, таких як ЦП, пам'ять, диск і мережа, необхідні для завершення емуляції. Одним із типів віртуальної машини є віртуальний брандмауер.

Віртуальний брандмауер є ідеальним рішенням для захисту віртуалізованого мережевого середовища. Три потенційні випадки використання віртуального брандмауера включають:

– Розгортання загальнодоступної хмари: організації все більше використовують розгортання публічної хмари, використовуючи такі служби, як AWS, GCP і Azure, для зберігання та обробки критичних даних. Віртуальний брандмауер є важливою частиною здатності організації захищати від кіберзагроз і відповідати вимогам відповідності в цих середовищах.

– Розгортання приватної хмари: віртуальні брандмауери також можуть бути цінними інструментами в приватних хмарних середовищах. Вони часто містять такі функції, як автоматичне надання, масштабованість і динамічне керування об'єктами та політиками, які спрощують безпеку в приватних хмарах.

– Розташування філій і програмно-визначені середовища: із зростанням програмно-визначених мереж (SDN) і програмно-визначених глобальних мереж (SD-WAN) корпоративна мережа все більше оптимізується та віртуалізується. Віртуальний брандмауер можна легко розгорнути на пристроях SD-WAN із вбудованим програмним забезпеченням гіпервізора, щоб допомогти перенести безпеку на край мережі.

Навіщо потрібен віртуальний брандмауер

Віртуальний брандмауер розроблено для забезпечення багатьох тих самих засобів захисту, що й традиційні фізичні пристрої брандмауера , але як хмарне рішення. Це дозволяє їм задовольнити кілька потреб безпеки:

– Перевірка руху з півночі на південь: хмарні ресурси розгортаються за межами традиційного периметра корпоративної мережі та доступні безпосередньо з загальнодоступного Інтернету. Розгортання пристрою віртуального брандмауера для перевірки та фільтрації вхідного та вихідного трафіку для цих хмарних ресурсів має важливе значення для захисту їх від компрометації та потенційного витоку даних.

– Перевірка трафіку «Схід-Захід». Навіть якщо організація контролює доступ до своїх хмарних ресурсів, перевірка потоків даних «Схід-Захід» у середовищі організації також є життєво важливим аспектом кібербезпеки. Кіберзлочинці, які мають доступ до мережі організації, зазвичай переміщуються через неї, щоб отримати доступ до конфіденційних ресурсів і досягти своїх кінцевих цілей. Зі зростаючим обсягом конфіденційних даних і функціональних можливостей, розгорнутих у хмарі, виконання інспекції вмісту та застосування політики безпеки цих потоків трафіку зі сходу на захід є важливими для захисту хмарних ресурсів, що робить віртуальний брандмауер необхідним.

– Розташування розгортання: зростаючий відсоток інфраструктури організації розгортається у віртуалізованих середовищах, таких як хмара. Захист цих середовищ за допомогою фізичного брандмауера часто не є життєздатним варіантом, оскільки ці пристрої не можна розгорнути на місці, а маршрутизація трафіку через мережу штаб-квартири для перевірки безпеки не є життєздатним варіантом. Хмарний або віртуальний брандмауер дозволяє організації розгортати той самий рівень безпеки у форм-факторі, який розроблено для середовища розгортання та добре підходить для нього.

– Гнучкість і масштабованість: віртуальні брандмауери зазвичай розгортаються як рішення безпеки в хмарних середовищах. Організації зазвичай

використовують хмару через її вбудовану гнучкість і масштабованість, тому безпека хмари також повинна мати можливість адаптуватися до мінливих вимог. З цієї причини використання віртуальних брандмауерів – потенційно через службу «Брандмауер як послуга » (FWaaS), що пропонує доступ до захисту за вимогою, – є ідеальним рішенням для захисту цих хмарних середовищ, особливо з можливістю автоматизації стандартних етапів розгортання, надання та налаштування.

Як працює віртуальний брандмауер

Віртуальні брандмауери зазвичай розгортаються як віртуальні машини в хмарному середовищі або через пропозицію FWaaS. Це дає змогу організації використовувати переваги гнучкості та масштабованості хмари також у своїй безпеці.

Як і будь-який брандмауер, віртуальний або хмарний брандмауер повинен мати можливість перевіряти трафік, що входить і виходить із захищеної мережі. Віртуальний брандмауер має кілька варіантів для цього:

– Режим мосту: віртуальний брандмауер можна розгорнути як фізичний брандмауер, який розташовується безпосередньо на шляху трафіку. Це дає йому змогу перевіряти та дозволяти або блокувати будь-який трафік, який намагається увійти або вийти з віртуального середовища через міст.

– Власні хмарні API: багато хмарних служб пропонують API, як-от AWS VPC Traffic Mirroring, який забезпечує видимість потоків трафіку в розгортанні хмари організації. Віртуальні брандмауери також можуть використовувати переваги цього віртуального мережевого відводу для перевірки трафіку, що надходить і виходить із захищеного віртуального середовища.

Ця видимість дозволяє хмарному брандмауеру застосовувати свої інтегровані політики безпеки та будь-які вбудовані можливості безпеки, наприклад аналіз підозрілого вмісту в ізольованому програмному середовищі. Залежно від налаштувань розгортання та конфігурації, брандмауер також можна налаштувати для блокування спроб атак або створення сповіщень.

Різні типи віртуальних брандмауерів можуть мати додаткові функції, які роблять їх ідеальними для захисту хмарних середовищ. Наприклад, використання Check Point динамічних об’єктів дає змогу визначати політики безпеки таким чином, щоб певні значення вирішувалися по-різному кожним шлюзом, який використовує політику. Це дає змогу визначати загальні політики безпеки, які послідовно застосовуються в усій IT-інфраструктурі організації та мають конкретні значення, наприклад IP-адреси, які встановлюються на основі інтеграції брандмауера з тегами хмарних програм.

Виробничі мережі стикаються з унікальними проблемами, і не кожен брандмауер створений для їх вирішення. Багато брандмауерів підприємств переповнені непотрібними функціями та застарілою складністю – це найгірший кошмар IT-менеджера. Давайте зосередимося на тому, що насправді потрібно виробникам у брандмауері для 2025 року.

Що шукати в брандмауері

1. Простота

Брандмауер має захищати вашу мережу, а не ускладнювати її. Надто складні системи з роздутими інтерфейсами керування та незрозумілими налаштуваннями призводять до неправильної конфігурації, що є основною причиною зламів.

Виберіть брандмауер, який надає перевагу простоті. Від рішень для малих підприємств, таких як Meraki або Fortinet , до більш передових технологій, таких як MikroTik , і навіть орієнтованих на особисте використання, таких як PFsend або Firewalla – рішень, які пропонують потужну функціональність рівня 3 (L3) без надмірностей.

Ключовий висновок: якщо ваш брандмауер потребує команди консультантів для належного налаштування, він вас уже підвів.

2. Рівень захисту 3

Захист рівня 3 є основою для будь-якого брандмауера. Він створює перший внутрішній і зовнішній шар захисту, який потрібен кожному виробнику.

Фільтрація L3 дає змогу керувати трафіком на основі IP-адрес (і портів, хоча постачальники брандмауерів люблять неправильно називати свій інтерфейс і говорити «протоколи»…) – без жодних помилок, лише ефективний контроль периметра.

Ключовий висновок: створіть потужний перший бар'єр – забороніть усі вхідні – без надмірного проектування. Ця перша перешкода тут, щоб зменшити 98% ризиків низької складності.

3. Безпека: швидко оновлюйте та додайте ще один рівень захисту від іншого постачальника

Ось сувора правда: деякі постачальники брандмауерів пережили кілька важких років, нагромадивши CVE (поширені вразливості та ризики), які поставили своїх клієнтів під загрозу. Наша рекомендація полягає в тому, щоб запровадити другий вбудований брандмауер від іншого постачальника, щоб радикально зменшити ризик експлойтів, як ми бачили все літо 2024 року на шлюзі Fortinet.

Тут також приходить простота: якщо обидва брандмауери прості, ви перебуваєте в хорошій позі, якщо обидва складні, ви не встигнете вдома до обіду. Тут також можна впроваджувати рішення, подібні до DMZ, cough cough Trout, щоб захистити від цього ризику.

4. Рівень захисту 4-7

Оскільки нині більшість трафіку шифрується, лише L3 не впорається. Вам слід звернути увагу на вищий захист рівня OSI.

Наша рекомендація? Шукайте проксі-сервери, щоб додати потужний захисний бар'єр на вищих рівнях і забезпечити видимість. На жаль, RSA та позасмуговий аналіз припиняються, а для конфіденційних ресурсів краще використовувати вбудовані проксі. Програмні рішення, такі як програмно визначений повітряний зазор, можуть забезпечити необхідну масштабованість.

Чому виробничі мережі особливі

Виробники стикаються з такими проблемами, як брандмауери загального призначення:

1. Більші об’єкти: виробничі потужності зазвичай вимагають більше площі, ніж послуги, з обладнанням і пристроями, розташованими у великих будівлях або кількох будівлях.

2. Обладнання сторонніх виробників : від верстатів з ЧПК до систем HVAC, виробники значною мірою покладаються на пристрої сторонніх виробників, багато з яких потребують віддаленого доступу для обслуговування.

3. Спільні IT-пристрої: працівники виробництва часто використовують спільні комп’ютери або термінали, що ускладнює відстеження та автентифікацію дій користувачів.

Практичні рекомендації щодо брандмауера для виробників

1. Почніть Simple з периметра

Встановіть простий брандмауер одразу після шлюзу провайдера. Основною роботою тут є базова фільтрація вхідних і вихідних повідомлень. Вам не потрібні складні налаштування – лише правила, які визначають, що надходить, а що виходить.

Важлива порада: вимкніть доступ до керування через інтерфейс WAN. Це одна з найпоширеніших і небезпечних неправильних конфігурацій, які ми все ще бачимо у виробничих мережах.

2. Використовуйте маршрутизовані локальні мережі для внутрішньої сегментації

Вважайте свою внутрішню мережу ненадійною. Комбінуйте VLAN для таких пристроїв, як точки доступу Wi-Fi, з маршрутизованими локальними мережами для внутрішнього зв’язку.

– Чому маршрутизовані локальні мережі? Вони забезпечують кращу безпеку (сегментація), видимість (аналіз IP джерела/приймача) і продуктивність (без широкомовних штормів).

– Одна VLAN для вашого Wi-Fi : створіть VLAN, щоб перегрупувати вашу точку доступу та забезпечити постійне підключення під час роумінгу в будівлі.

– Потім масштабуйте за допомогою маршрутизованих локальних мереж . Використовуйте механізми маршрутизації та рівня 3, щоб увімкнути зв'язок зі сходу на захід у вашій мережі – будь то комп'ютер, що під'єднується до принтера, комп'ютер, що взаємодіє з системою ERP, або ПЛК, який зв'язується зі SCADA.

3. Автентифікація користувачів і машин

Спільні IT-пристрої поширені на виробництві, але вони створюють сліпі зони безпеки. Ви повинні знати, *хто* має доступ до ваших систем, а не тільки *що* .

Ось як:

– Увімкніть особистий вхід : якщо ви використовуєте такі платформи, як Microsoft 365 або Google Workspace, запровадьте вхід окремих працівників для спільних пристроїв.

– Інтеграція з постачальниками ідентифікаційної інформації: централізоване керування ідентифікаційною інформацією спрощує відстеження та контроль доступу, забезпечуючи підзвітність у вашій мережі.

– Використовуйте рішення фізичних карт-ключів для автентифікації робочої сили в рукавичках і окулярах.

3.2 Розробка структурної схеми

За усе більше широким застосуванням технологій віртуалізації набирають популярність віртуальні пристрої безпеки (Virtual Security Appliance, VSA). По суті, у випадку VSA можна говорити про зворотний перехід на рівень ПЗ.

Віртуальні шлюзи безпеки

Багато шлюзів безпеки можуть установлюватися не тільки на фізичних, але й на віртуальних серверах (VMware, XenServer, Microsoft Windows Server з Hyper-V) або розробляються спеціально для віртуального середовища. Нерідко

вони реалізуються на рівні гіпервізора й застосовуються для забезпечення безпеки усередині самого віртуального середовища, контролюючи трафік між віртуальними машинами (ВМ). Це відносно новий і швидко, що розвивається сегмент.

Наприклад, інтернет-шлюзи eSafe компанії Aladdin сертифіковані VMware, і будь-який бажаючий може завантажити ВМ із передвстановленим eSafe з відповідного розділу сайту VMware. Trend Micro застосовує в продуктах підхід Software Virtual Appliance, що залишає замовникові право вибору найбільш актуальних для нього переваг фізичного або віртуального рішення. Практично кожний продукт Trend Micro можна розгорнути на стандартному фізичному сервері (з образа ISO) або як Virtual Appliance для віртуального середовища VMware ESX або Infrastructure. У першому випадку замовник одержує настроєний програмно-апаратний комплекс із захищеної ОС і продуктом Trend Micro (Software Appliance).

Продукти компанії Check Point передбачають, крім іншого, реалізації для VMware і можуть контролювати трафік між ВМ, не пропускаючи його через зовнішній комутатор. Компанія Barracuda оголосила в липні про плани випуску в найближчі місяці повної лінійки віртуалізованих систем, починаючи із продуктів Spam & Virus Firewall Vx і SSL VPN Vx. Замовники зможуть здобувати їх у різних варіантах, включаючи традиційні й віртуальні пристрої безпеки, SaaS і гібридні версії.

У випадку VSA мова йде про два сценарії:

– Перший – застосування на одній апаратній платформі декількох віртуальних засобів захисту, що дозволяють контролювати певне число мережних сегментів з різними правилами доступу й керуванням, завдяки чому вдається заощадити на вартості засобів захисту.

– Другий сценарій має на увазі захист самих віртуалізованих інфраструктур з урахуванням їх особливостей, однак реалізація такого рішення – дуже непросте завдання.

Якщо розглядати проблему з погляду форм-фактора, то майбутнє – за віртуальними пристроями. У міру розгортання часток «хмар» переваги VSA почнуть здобувати все більшу значимість. Завдяки віртуальному форм-факторові покупці зможуть вибирати кращі у своєму класі рішення й не погоджуватися на компроміс у вигляді UTM. Однак VSA – лише ще один тип продуктів ІБ, причому його одного недостатньо: необхідний контроль трафіку як між віртуальними, так і між фізичними серверами. Наприклад, спільне рішення Juniper і Altor Networks дозволяє організувати комплексний захист віртуальних середовищ із використанням традиційних апаратних засобів забезпечення ІБ і VSA.

Trend Micro найближчим часом планує випустити Deep Security 7.5 – продукт для захисту фізичних, віртуальних і «хмарних» систем. У новій версії захисні модулі IDS/IPS, міжмережного екрана, контролю цілісності й аналізу журналів подій доповнені модулем антивірусного захисту для ВМ. Антивірусна перевірка здійснюється через VMware API VMsafe (Seraph) на рівні гіпервізора й працює без установки агента у ВМ. А версія, що вийшла наприкінці серпня, Office Scan 10.5 покликана вирішити проблеми продуктивності при антивірусному захисті в середовищі VDI.

IBM ISS пропонує рішення для захисту ВМ «зовні й зсередини»: IBM Proventia Server IPS і IBM Virtual Server Security for VMware vSphere, де використовується інтерфейс VMware VMsafe. Деякі вендори розглядають VSA лише як перше покоління засобів захисту віртуального середовища, указуючи на необхідність більше глибокої інтеграції функцій безпеки в середовище віртуалізації.

Тим часом, більше 70% організацій різних форм власності й розмірів уже розгорнули VSA або планують зробити це. Основними факторами росту даного ринку служать збільшення кількості погроз безпеки, швидке поширення віртуалізації серверів і нові, специфічні для віртуального середовища, проблеми ІБ.

На рисунку 3.1 зображена структурна схема роботи системи. Схема розділена на три основних компоненти:

– шлюз інформаційної безпеки при роботі з хмарою з використанням технології VSA;

– розроблена програма;

– користувач.

Коли користувач підключає персональний комп'ютер до шлюзу інформаційної безпеки при роботі з хмарою з використанням технології VSA, відбувається розпізнання операційною системою типу пристрою й виводу меню вироблених дій.

Розроблена програма перехоплює системне повідомлення операційній системі про виклик меню вироблених дій над шлюзом інформаційної безпеки при роботі з хмарою з використанням технології VSA і активізує власний інтерфейс програми.

Розроблена програма складається з декількох частин:

– інтерфейсу програми;

– модуля потокового шифрування RC4;

– модуля потокового дешифрування RC4;

– налаштування програми й захисту програми.

Розроблена програма управляє процесом обміну інформацією між шлюзом інформаційної безпеки при роботі з хмарою з використанням технології VSA і персональним комп'ютером, використовуючи потоковий алгоритм шифрування інформації RC4. Завдяки такому підходу, можливо використовувати всі існуючі на даний момент шлюзи інформаційної безпеки при роботі з хмарою з використанням технології VSA не зупиняючись на окремих реалізаціях з підвищеними вимогами захищеності шлюза інформаційної безпеки при роботі з хмарою з використанням технології VSA (рисунок 3.1).

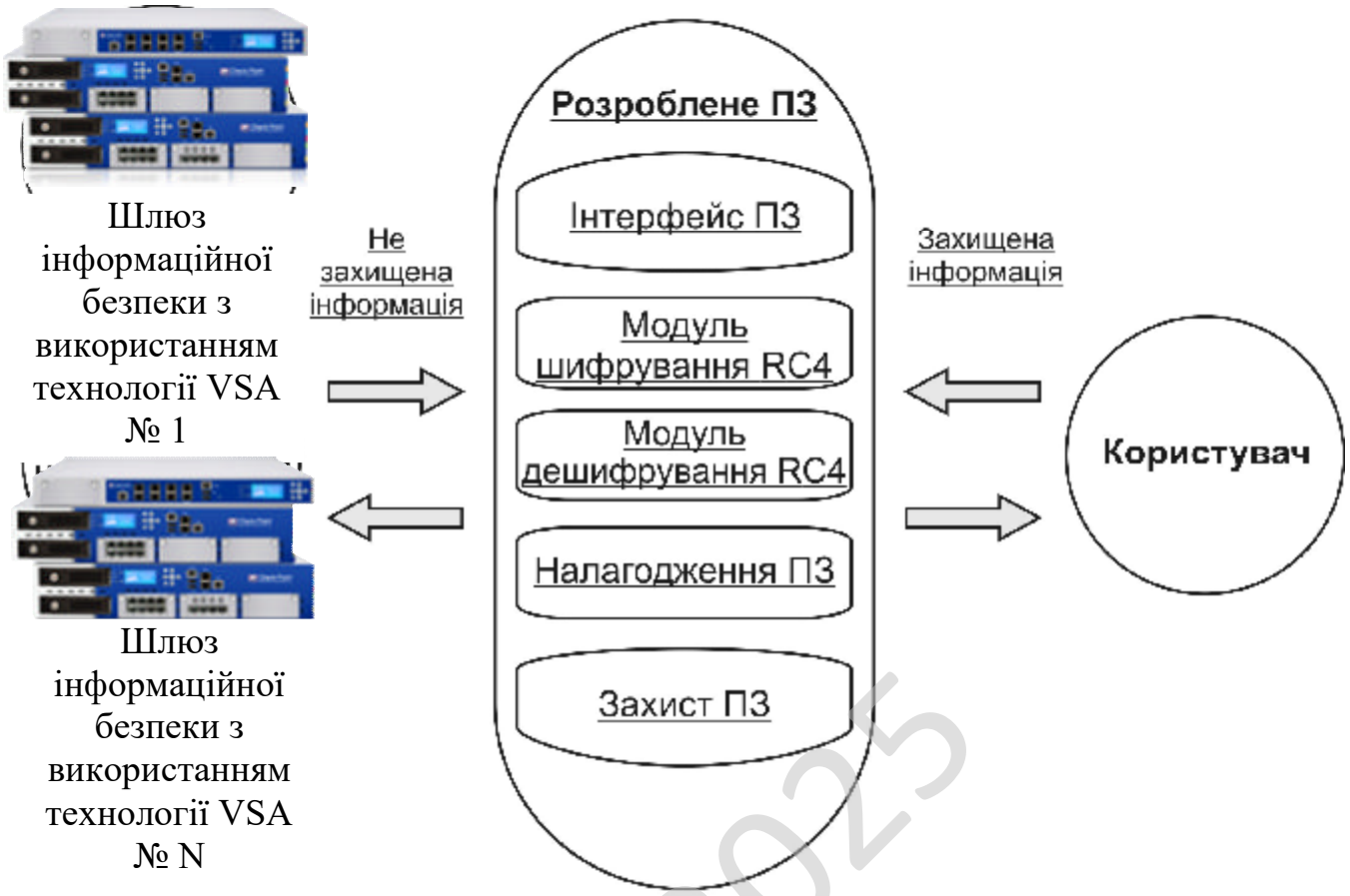

Рисунок 3.1 – Структурна схема роботи системи

3.3 Розробка функціональної схеми

У якості криптоалгоритму спрямованого на захист інформації, яка знаходиться у хмарі поза шлюзом інформаційної безпеки з використанням технології VSA візьмемо алгоритм RC4. Розглянутий нами криптоалгоритм RC4 відноситься до класу потокових шифрів, які останнім часом стали популярними завдяки високій швидкості роботи. Потокові шифри перетворять відкритий текст у шифротекст по одному біті за операцію. Генератор потоку ключів (іноді називаний генератором із ключем, що біжить) видає потік біт: k_1, k_2, k_3, ..., k_i. Цей потік ключів і потік біт відкритого тексту, p_1, p_2, p_3, ..., p_i, піддаються операції “або, що виключає", і в результаті виходить потік біт шифротексту.

$$c_i = p_i \oplus k_i \qquad (3.1)$$

При дешифруванні операція XOR виконується над бітами шифротексту й тим же самим потоком ключів для відновлення біт відкритого тексту.

$$p_i = c_i \oplus k_i \quad (3.2)$$

Безпека системи повністю залежить від властивостей генератора потоку ключів. Генератор потоку ключів створює бітовий потік, що схожий на випадковий, але в дійсності детермінований і може бути безпомилково відтворений при дешифруванні. Чим ближче вихід генератора потоку ключів до випадкового, тим більше часу буде потрібно для взлому шифру.

Для всіх потокових шифрів використовуються ключі. Вихід генератора потоку ключів є функцією ключа. Тепер, якщо одержати пару відкритий текст/шифротекст, то можна читати тільки ті повідомлення, які зашифровані тим же ключем. Потокові шифри особливо корисні для шифрування нескінченних потоків комунікаційного трафіку, наприклад, при записі даних у шлюз інформаційної безпеки з використанням технології VSA .

Генератор потоку ключів складається із трьох основних частин:

– Внутрішній стан описує поточний стан генератора потоку ключів.

– Два генератори потоку ключів, з однаковим ключем і однаковим внутрішнім станом, видають однакові потоки ключів.

– Функція виходу по внутрішньому стану генерує біт потоку ключів.

– Функція наступного стану по внутрішньому стану генерує новий внутрішній стан.

Криптоалгоритм RC4 відноситься до так званих шифрів, що самосинхронізуються. У потокових шифрах, що самосинхронізуються, кожний біт потоку ключів є функцією фіксованого числа попередніх біт шифротексту. Військові називають цей шифр автоключом шифротексту.

Потоковий шифр, що самосинхронізується, показаний на рисунку 3.3. Внутрішній стан є функцією попередніх n біт шифротексту. Криптографічно складною є вихідна функція, що використовує внутрішній стан для генерації біта потоку ключів.

Так як внутрішній стан повністю залежить від попередніх *n* шифротексту, дешифруючий генератор потоку ключів автоматично синхронізується з генератором, що шифрує, потоку ключів, прийнявши *n* біт шифротексту. В інтелектуальних реалізаціях цього режиму кожне повідомлення починається випадковим заголовком довжиною *n* біт.

Цей заголовок шифрується, передається й потім розшифровується. Розшифровка буде неправильною, але після цих *n* біт обидва генератори потоку ключів будуть синхронізовані.

Слабкою стороною потокового шифру, що самосинхронізується, є поширення помилки. Для кожного біта шифротексту, зіпсованого при передачі, дешифруючий генератор потоку ключів видає n неправильних біт потоку ключів. Отже, кожному неправильному біту шифротексту відповідають n помилок у відкритому тексті, поки зіпсований біт не перестане впливати на внутрішній стан.

Алгоритм RC4 і його криптоаналіз

Істотне підвищення продуктивності мікропроцесорів в 80-і роки викликало в криптографії посилення інтересу до програмних методів реалізації криптоалгоритмів як можливій альтернативі апаратним схемам на регістрах зрушення.

Одним з найперших подібних криптоалгоритмів, що получили широке поширення, став RC4. Алгоритм RC4 – це потоковий шифр зі змінною довжиною ключа.

Він володіє наступними властивостями:

– адаптивністю для апаратних засобів і програмного забезпечення, що означає використання в ньому тільки примітивних обчислювальних операцій, звичайно присутніх на типових мікропроцесорах;

– алгоритм швидкий, тобто в базисних обчислювальних операціях оператори працюють на повних словах даних;

– адаптивністю на процесори різних довжин слова;

– компактністю в термінах розміру коду, і особливо зручний для процесорів з побайтно-орієнтованою обробкою;

– низькою вимогою до пам'яті, що дозволяє реалізовувати алгоритм на пристроях з обмеженою пам'яттю;

– використанням циклічних зрушень, залежних від даних, з "змінним" числом;

– простотою й легкістю виконання.

У цей час алгоритм RC4 реалізований у десятках комерційних криптографічних продуктів, включаючи Lotus Notes, Apple Computer's AOCE, Oracle Secure SQL, а також є частиною специфікації стандарту стільникового зв'язка CDPD.

Криптогенератор функціонує незалежно від відкритого тексту. Генератор має підстановочну таблицю (S-бокс 8 x 8): S_0, S_1, . . ., S_{255}. Входами генератора є замінені по підстановці числа від 0 до 255, і ця підстановка є функцією від ключа змінюваної довжини. Генератор має два лічильники i і j, ініціалізуємих нульовим значенням.

Для генерації випадкового байта гами виконуються наступні операції:

$$i = (i+1) \bmod 256 \quad (3.3)$$

$$j = (j+S_i) \bmod 256 \quad (3.4)$$

$$\text{swap}\ (S_i, S_j) \quad (3.5)$$

$$t = (S_i+S_j) \bmod 256 \quad (3.6)$$

$$K = S_t \quad (3.7)$$

Байт K складається операцією XOR з відкритим текстом для виробітку шифротексту, або із шифротекстом для одержання байта відкритого тексту. Шифрування відбувається досить швидко – приблизно в 10 разів швидше DES-алгоритму. Ініціалізація S-боксу настільки ж проста. На першому кроці він заповнюється лінійно: $S_0 = 0$, $S_1 = 1$, . . ., $S_{255} = 255$.

Потім ще один 256-байтний масив повністю заповнюється ключем, для чого ключ повторюється відповідне число раз залежно від довжини: K_0, K_1,...,K_{255}. Індекс *j* обнуляется. Потім:

```
for (i=0; i<= 255; i++)
{
     j = (j+Si+Ki) mod 256;
     swap (Si , Sj);
}
```

Схема показує, що RC4 може приймати приблизно 2^{1700} (256! * 256^2) можливих станів. *S*-бох повільно змінюється в процесі роботи: параметр *i* забезпечує зміну кожного елемента, а *j* відповідає за те, щоб ці елементи змінювалися випадковим образом.

Фактично, RC4 являє собою сімейство алгоритмів, що задаються параметром *n*, що є позитивним цілим з рекомендованим типовим значенням $n = 8$.

Внутрішній стан генератора RC4 у момент часу t складається з таблиці $S_t = (S_t(L))_{t=0}^{n2^n-1}$, що містить 2^n n-бітних слів і із двох n-бітних слів-покажчиків i_t і j_t. Таким чином, розмір внутрішньої пам'яті становить $M = n2^n + 2n$ біт. Нехай вихідне *n*-бітне слово генератора в момент *t* позначається як Z_t.

Нехай початкові значення $i_0 = j_0 = 0$. Тоді функція наступного стану й функція виходу RC4 для кожного $t > 1$ задається наступними співвідношеннями:

$$i_t = i_{t-1} + 1 \quad (3.8)$$

$$j_t = j_{t-1} + S_{t-1}(i_t) \quad (3.9)$$

$$S_t(i_t) = S_{t-1}(j_t) \quad (3.10)$$

$$S_t(j_t) = S_{t-1}(i_t) \quad (3.11)$$

$$Z_t = S_t(S_t(i_t) + S_t(j_t)), \quad (3.12)$$

де всі додавання виконуються по модул. 2^n. Мається на увазі, що всі слова, крім тих, які піддаються перестановці, залишаються тими ж самими. Вихідна послідовність n-бітних слів позначається як $Z_t = (Z_t)_{t=1}^{\infty}$.Початкова таблиця S_0 задається в термінах ключової послідовності:

$$K = (K_L)_{t=0}^{2n-1} \quad (3.13)$$

з використанням тієї ж самої функції наступного стану, починаючи від таблиці одиничної підстановки $(L)^{2n}{}_{L=0}{}^{2n-1}$. Більш строго, нехай $j_0 = 0$ і для кожного $1 \le t \le 2^n$ обчислюється $j_t = (j_t - 1 + S_{t-1}(t\text{-}1) + K_{t-1}) \bmod 2n$, а потім переставляються місцями $S_{t-1}(t\text{-}1)$ і $S_{t-1}(j_t)$.

На останньому кроці породжується таблиця, що представляє S_0. Ключова послідовність K складається із секретного ключа, що можливо повторюється, і випадкового ключа, переданого у відкритому виді з метою ресинхронизації.

До останнього часу у відкритій літературі практично не було публікацій по криптоаналізу алгоритму RC4. Компанія RSA Data Security оголосила, що шифр має імунітет до методів лінійного й диференціального криптоаналізу, високо не лінієн і не схоже, щоб у нього були короткі цикли.

Відзначається, що для послідовностей, генеруємих RC4, не підходять методи статистичного аналізу. Але, з іншого боку, для блоків, розмір яких перевищує M (розмір внутрішньої пам'яті генератора), завжди існує лінійна статистична слабість або так звана "лінійна модель". Таку модель можна ефективно визначати за допомогою методу апроксимації лінійною послідовною схемою. Лінійна статистична слабість – це лінійне співвідношення між бітами гами, що виконується з імовірністю, що відрізняється від 1/2.

За допомогою методу АЛПС були виведені лінійні моделі для RC4. Метод АЛПС полягає в знаходженні й рішенні послідовної лінійної схеми, що апроксимує генератор гами й приводить до лінійних моделей з відносно великим кореляційним коефіцієнтом c, де ймовірність відповідного лінійного співвідношення між бітами гами становить $(1 + c)/2$. При аналізі використовувалася техніка двійкових похідних. Нехай $Z = (Z_t)_{t=1}^{\infty}$ позначає послідовність самих молодших біт слів виходу RC4, і нехай $Z' = (Z'_t = Z_t + Z_{t+1})_{t=1}^{\infty}$ і $Z'' = (Z''_t = Z_t + Z_{t+2})_{t=1}^{\infty}$ позначають її перші й другу двійкові похідні, відповідно. Показано, що Z' не корелює ні з 1, ні з 0, але Z'' корелює з 1 з кореляційним коефіцієнтом, близьким до $15^* 2\text{-}3^n$ при великих $2n$, де n – довжина ключа.

Оскільки довжина вихідної послідовності, необхідна для виявлення статистичної слабості з кореляційним коефіцієнтом c, становить O(c-2), те ця довжина дорівнює приблизно 64^n /225. Наприклад, якщо $n = 8$, як рекомендується в більшості додатків, те необхідна довжина близька до 2^{40}.

Результати комп'ютерних експериментів погодяться з теоретичними пророкуваннями. Оскільки результуючий коефіцієнт кореляції істотно перевищує величину $2^{M/2}$, те встановлену лінійну модель варто розглядати як статистичну слабість генератора, принаймні в теоретичному аспекті.

Був проведений криптоаналіз узагальненої схеми вузла, що комбінує, з довільним розміром пам'яті. Досліджено кореляційні властивості таких вузлів, обґрунтовані нові конструктивні критерії, пропоновані до схем подібного типу. Розроблено ефективний метод апроксимації лінійною послідовною схемою для побудови лінійних функцій від входу й виходу з порівняно більшим коефіцієнтом взаємної кореляції. Це практична процедура, що дозволяє з високою ймовірністю знаходити пари взаємно коррельованих лінійних функцій (від якнайбільше $M + 1$ послідовних вихідних біт і якнайбільше $M + 1$ послідовних векторів входу) з порівняно більшими коефіцієнтами кореляції. Метод АЛПС складається в завданні й рішенні лінійної послідовної схеми (ЛПС), що апроксимує вузол, що комбінує, з пам'яттю. Ця ЛПС має додаткові незбалансовані входи й заснована на лінійних апроксимаціях функції виходу й всіх компонентів функції наступного стану. Лінійна апроксимація булевої функції – це будь-яка лінійна функція, з якою задана булева функція скорельована. Описаний метод застосуємо до довільних вузлів, що комбінують, з пам'яттю без обмежень на функції виходу й наступний стан.

Спочатку відшукуються лінійні апроксимації функції виходу f і кожної з функцій-компонентів функції наступного стану F. Це еквівалентно вираженню кожної із цих $M + 1$ функцій у вигляді суми лінійної функції й незбалансованої функції. Якщо підлягаючої декомпозиції функція вже несбалансована, то можна вибрати константно-нульову лінійну функцію. Якщо підлягаюча декомпозиції

функція статистично незалежна від деякої підмножини змінних, то кожна лінійна апроксимація з необхідністю повинна задіяти принаймні одну зі змінних цієї підмножини. Основна вимога – щоб відповідні кореляційні коефіцієнти відрізнялися від нуля. Також бажано, щоб вибиралися лінійні апроксимації з кореляційними коефіцієнтами, абсолютні значення яких близькі до максимального. Кореляційні коефіцієнти можна визначати за допомогою техніки перетворення Уолша.

На наступному кроці, одержавши лінійні апроксимації, у матричній формі записують базові рівняння вузла, що комбінує, з пам'яттю

$$S_{t+1} = A \cdot S_t + B \cdot X_t + \Delta(X_t, S_t),\ t \geq 0, \tag{3.14}$$

$$y_t = C \cdot S_t + D \cdot X_t + \varepsilon(X_t, S_t),\ t \geq 0, \tag{3.15}$$

де вектори розглядаються як матриці-стовпці; A, B, C, D – двійкові матриці; а ε і кожний компонент в $D = (d_1, \ldots, d)$ – незбалансовані булеві функції, іменовані функціями шуму. Основна ідея полягає в тому, щоб розглядати $\{\varepsilon(X_t, S_t)\}_{t=0}^{\infty}$ і $\{\delta(X_t, S_t)\}_{t=0}^{\infty}$, $1 \leq i \leq M$, як вхідні послідовності, так що останні рівняння виявляються які задають неавтономну лінійну машину з кінцевим числом станів або ЛПС, іменовану АЛПС вузла, що комбінує, з пам'яттю. Тоді можна вирішувати цю ЛПС із використанням техніки виробляючих функцій (D-перетворень). Зокрема, нехай S, X, Δ, ε, у позначають виробляючі функції від змінної z для послідовностей $\{S_t\}$, $\{X_t\}$, $\Delta(X_t, S_t)\}$, $\varepsilon(X_t, S_t)\}$, $y_t\}$, відповідно. Тоді рівняння зводяться до виду:

$$y = \left(D - \frac{C \cdot adj(zA - I)B}{\det(zA - I)} \right) X - \frac{C \cdot adj(zA - I)}{\det(zA - I)} (z\Delta + S_0) + \varepsilon \tag{3.16}$$

де I – одинична матриця, $\det(z - I) = \phi(z)$, $\phi(0) = 1$, – багаточлен, зворотний до характеристичного багаточлена матриці переходів A ступеня, що не перевищує ранг A ($\leq M$); а елементи (приєднаної) матриці $\mathrm{adj}(z - I)$ – це поліноми від z ступеня не більше M-1. Обчислювальна складність для відшукання такого рішення становить $O(M^3(N+1))$. В іншому виді рішення можна переписати як

$$y = \frac{1}{\varphi(z)} \sum_{i=1}^{N} g_i(z) x_i + \frac{1}{\varphi(z)} \sum_{j=1}^{M} h_j(z)(z\delta_j + s_{j0}) + \varepsilon \qquad (3.17)$$

де x_i і δ_j позначають виробляючі функції для $\{x_{it}\}$ і $\{\delta_j(X_t, S_t)\}$, а ступеня поліномів $g_i(z)$ і $h_j(z)$ якнайбільше рівні M і M-1, $1 \le i \le N$, $1 \le j \le M$, відповідно. Взявши

$$\varphi(z) = \sum_{k=0}^{M} \varphi_k z^k, \quad g_i(z) = \sum_{k=0}^{M} g_{ik} z^k, \quad h_j(z) = \sum_{k=0}^{M-1} h_{jk} z^k \qquad (3.18)$$

рішення можна перетворити до виду:

$$\sum_{k=0}^{M} \varphi_k y_{l-k} = \sum_{i=1}^{N} \sum_{k=0}^{M} g_{ik} x_{i,l-k} + e\left(X_i^{M+1}, S_{l-M}\right), \qquad t \ge M, \qquad (3.19)$$

$$e\left(X_i^{M+1}, S_{l-M}\right) = \sum_{j=1}^{N} \sum_{k=0}^{M-1} h_{jk} \delta_j\left(X_{t-1-k}, S_{t-1-k}\right) + \sum_{k=0}^{M-1} \varphi_k \varepsilon\left(X_{t-k}, S_{t-k}\right), \qquad t \ge M, \qquad (3.20)$$

де мається на увазі, що вектор стану S_{t-k} – це функція від ($X_{t-k-1}{}^{M-k}$, S_{t-M}) для кожного $0 \le k \le M$-1. Лінійні функції входу й виходу в (3.19) скоррельовані тоді й тільки тоді, коли функція шуму e незбалансована. Коефіцієнт кореляції не залежить від часу, якщо функція наступного стану збалансована. Якщо ця умова не задовольняється, то кореляційний коефіцієнт може залежати від часу, оскільки від S_t більше не потрібна збалансованість для кожного $t \ge 0$. Функція шуму e в (3.31) визначена як сума індивідуальних шумових функцій, які незбалансовані за умови, що збалансовано функцію наступного стану. Оскільки від індивідуальних шумових функцій не потрібно бути незалежними, у принципі не можна виключати можливість, що коефіцієнт кореляції e з константною нульовою функцією дорівнює нулю або дуже близький до цього значення.

У розглянутому випадку індивідуальні шумові функції можна трактувати як булеві функції від $n = MN + N + M$ змінних в ($X^{M+1}{}_t$, S_{t-M}). Отже, за винятком деяких особливих випадків, у загальному випадку можна з високою ймовірністю очікувати, що загальний кореляційний коефіцієнт дуже близький до добутку індивідуальних і, таким чином, відрізняється від нуля. Відповідно, метод АПЛС не тільки з високою ймовірністю дає взаємно коррельовані лінійні функції від входу й виходу, але також дозволяє оцінити значення відповідного кореляційного коефіцієнта, використовуючи незалежність або інші імовірнісні припущення.

Оскільки в ідеальному випадку хотілося б одержати такі АЛПС, у яких кореляційні коефіцієнти за абсолютним значенням близькі до максимуму, те індивідуальні кореляційні коефіцієнти повинні бути великими по величині, а кількість шумових членів в (3.20) повинне бути маленьким. Звичайно, ці вимоги можуть суперечити один одному. Тому гарним підходом буде повторення процедури АЛПС кілька разів, починаючи з найкращих лінійних апроксимацій для функції виходу й компонент функції наступного стану. Ця процедура може також виконуватися для всіх можливих лінійних апроксимацій, що представляється єдиним систематичним способом перевірити всі кореляції, виявлені в процесі застосування методу АЛПС. У загальному випадку є якнайбільше $(M+1)2^{M+N}$ таких лінійних апроксимацій. Однак, у принципі завжди можна перевірити всі можливі лінійні апроксимації навіть при великому M, оскільки в практичних реалізаціях функції виходу й наступного стану залежать від порівняно невеликої кількості змінних або ж складені з таких булевих функцій. Із практичної точки зору дана лінійна модель може бути використана для виділення по шифротексту генератора RC4 серед інших криптосистем, а також для відновлення параметра n. В 2000 році була опублікована стаття присвячена статистичному аналізу потокового генератора RC4, у якій були використані результати роботи для знаходження значення компонент S-боксу. Приблизний час роботи цього методу становить 2^{6n}, де n – порція біт у вихідному потоці, довжина вихідної послідовності, необхідна для виявлення статистичної слабості, близька до 2^{30}. Отриманий результат указує на істотну слабість генератора й можливість відновити параметри i і n. S-бокс може приймати 2^{nk}, де n_k – число біт ключа.

Розглянемо поетапно функціональну схему (рисунок 3.2). Функціональна схема – це схема, яка описує взаємодію й обробку даних. На функціональній схемі роботи системи представлений процес обробки інформації з шлюзів інформаційної безпеки при роботі з хмарою з використанням технології VSA з використанням алгоритму потокового шифрування RC4.

Інформація надходить із шлюза інформаційної безпеки при роботі з хмарою з використанням технології VSA в ядро алгоритму. Ядро алгоритму складається з функції генерації ключового потоку. Ця функція генерує послідовність біт, що потім поєднується з відкритим текстом за допомогою підсумовування по модулю два.

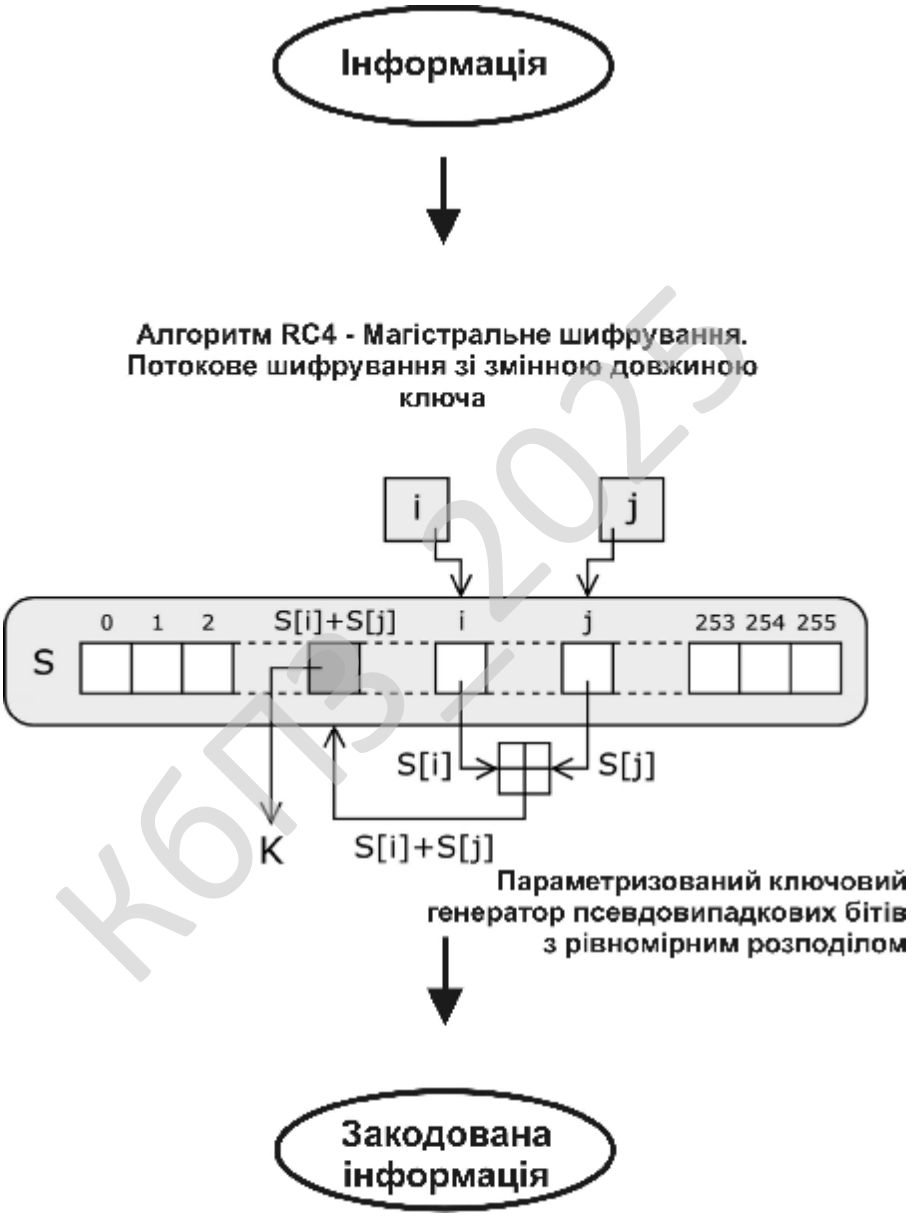

Рисунок 3.2 – Функціональна схема роботи системи

Процес дешифрації складається з регенерації цього ключового потоку й підсумовування його із шифрограммою по модулю два, відновлюючи вихідний текст. Також важливий елемент алгоритму функція ініціалізації, використовує ключ змінної довжини для створення початкового стану генератора ключового потоку.

Параметр n є розміром слова для алгоритму. За замовчуванням програма встановлює значення, $n = 8$. для підвищення рівня безпеки необхідно збільшити це значення.

Внутрішній стан RC4 складається з масиву S розміром 2^n слів і двох лічильників, кожний розміром в одне слово. Масив містить перестановку 2^n можливих значень слова. На плакаті два лічильники позначені через i і j.

Ключем як і іншими налаштуваннями алгоритму управляє користувач через меню налаштування програми.

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 дипломної роботи, наведена на рисунку 3.3.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Рисунок 3.3 – Діаграма взаємодії процесів

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

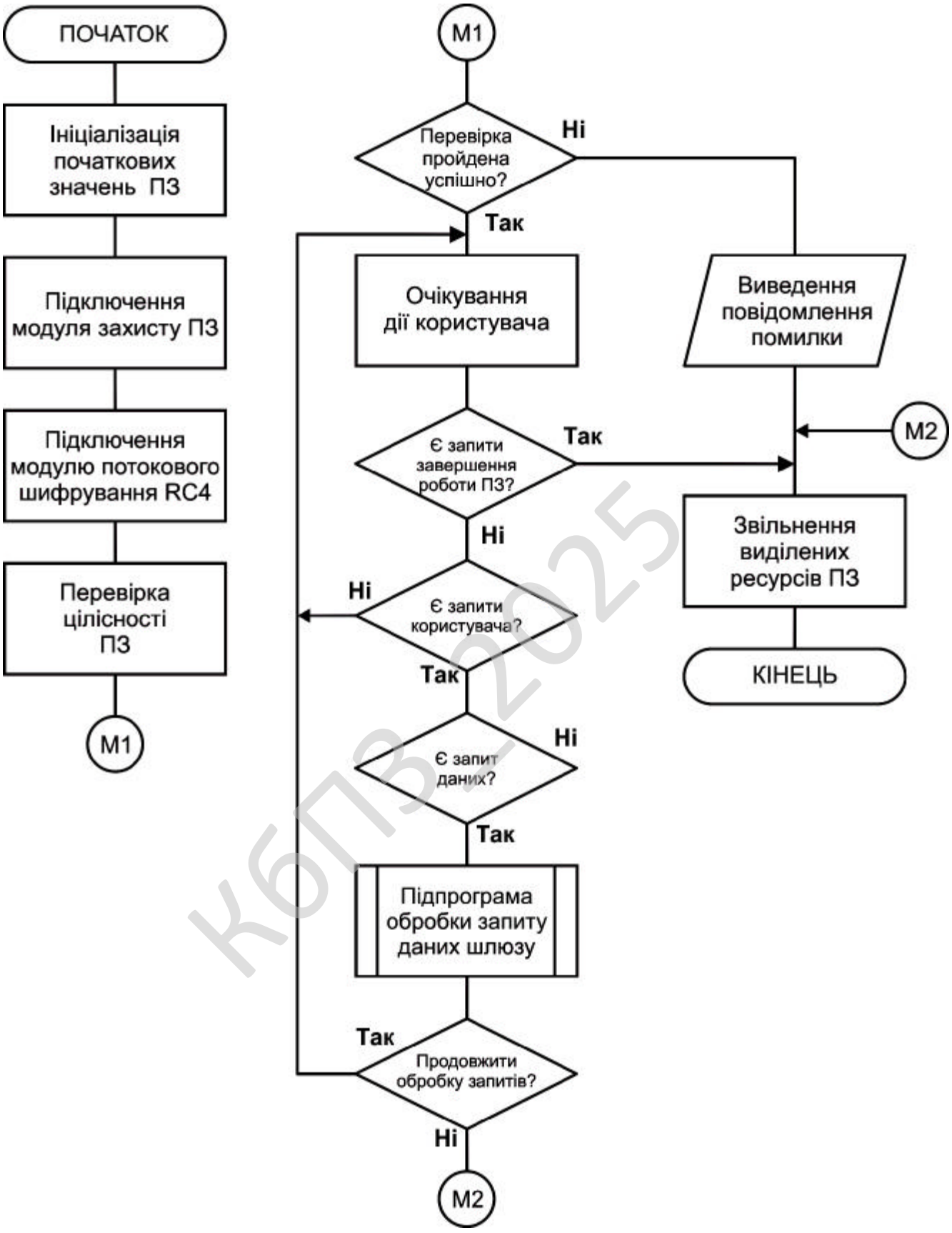

Рисунок 4.1 – Блок-схема основної програми

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

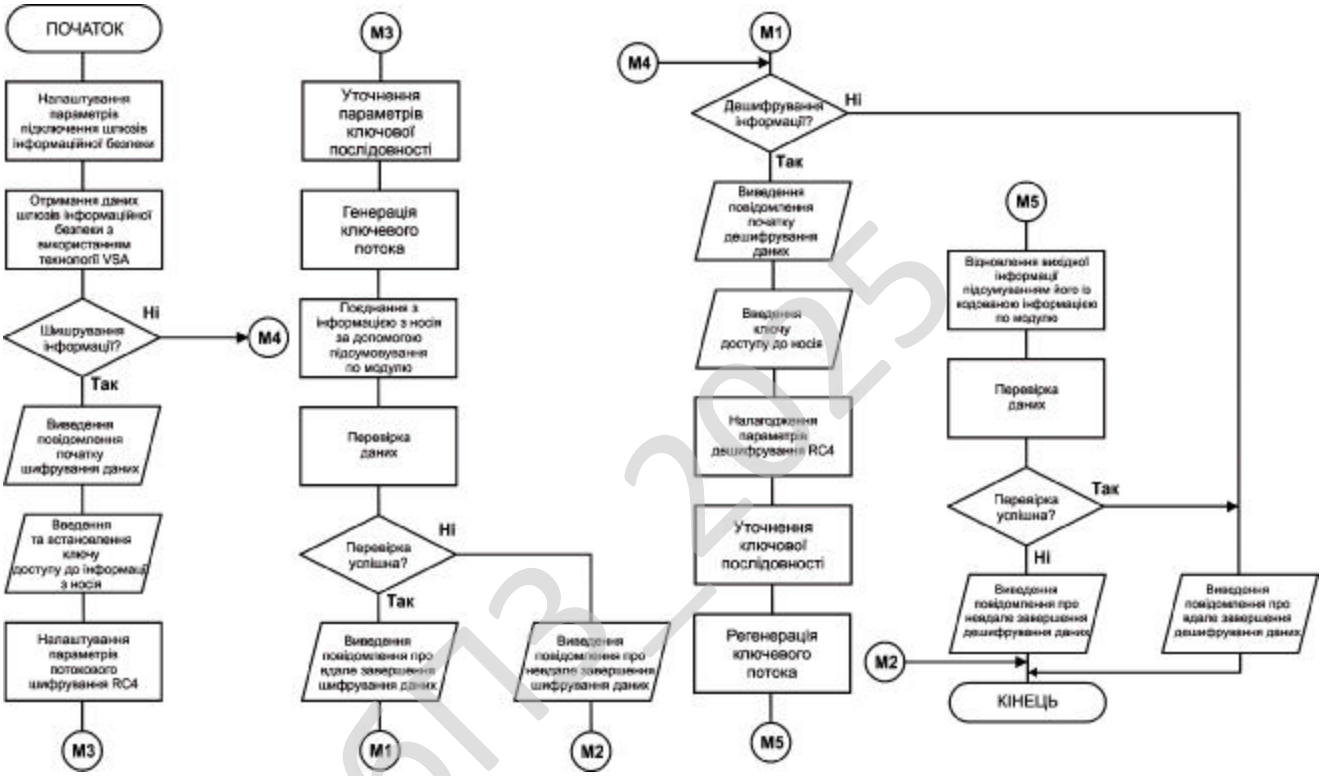

Рисунок 4.2 – Блок-схема роботи підпрограми

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем.

UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів; Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій.Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме — за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації

з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів. Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Опис системи

Система кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA забезпечує захист мережевого трафіку від несанкціонованого доступу, шкідливого програмного забезпечення та атак з боку зловмисників.

VSA (Virtual Security Appliance) реалізує функціональність мережевого захисту у віртуальному середовищі. Основні складові системи включають шлюз безпеки, модуль автентифікації, систему моніторингу трафіку, механізм аналізу загроз та засоби блокування підозрілої активності.

Шлюз безпеки є центральним елементом системи. Він приймає вхідний і вихідний трафік, аналізує його та застосовує правила безпеки. Фільтрація трафіку виконується за допомогою міжмережевого екрану, який використовує набір політик для запобігання несанкціонованому доступу.

Виявлення аномалій відбувається за рахунок аналізу поведінки пакетів. Автентифікація користувачів виконується за допомогою двофакторної перевірки, що включає перевірку облікових даних та одноразових кодів доступу.

Система моніторингу трафіку використовує алгоритми машинного навчання для виявлення аномальної активності. Вона зчитує потоки даних, аналізує частоту звернень, виявляє підозрілі шаблони та відправляє попередження адміністратору.

Аналіз загроз включає використання сигнатурного методу та поведінкового аналізу. Використання Python дозволяє інтегрувати сторонні

бібліотеки для роботи з мережевими протоколами та криптографічними алгоритмами.

Вихідний код реалізує основні функції захисту. Програма зчитує пакети з мережі, проводить їх аналіз та застосовує політики безпеки.

Основні модулі включають модуль обробки трафіку, модуль аналізу загроз та систему сповіщень. Обробка трафіку виконується через бібліотеку Scapy, що дозволяє аналізувати пакети, змінювати їх та створювати нові.

Аналіз загроз здійснюється через алгоритми, що використовують бібліотеку TensorFlow для розпізнавання аномальної поведінки.

Архітектура системи побудована за принципом розподіленої безпеки. Шлюз безпеки розгортається у віртуальному середовищі, що дозволяє масштабувати систему відповідно до навантаження. Система моніторингу функціонує на окремому сервері, що збирає дані про активність у мережі та передає їх до модуля аналізу загроз. Використання асинхронної обробки дозволяє зменшити затримки у виявленні атак.

Розрахунки підтверджують ефективність системи це середній час обробки одного пакета становить 0.3 мілісекунди, що дозволяє забезпечити захист без значного впливу на продуктивність мережі. Пропускна здатність шлюзу становить 10 Гбіт/с, що відповідає вимогам для корпоративних мереж. Використання машинного навчання збільшує точність виявлення загроз на 98.5% порівняно з традиційними методами.

Обрана архітектура дозволяє ефективно протидіяти загрозам та забезпечує масштабованість системи. Впровадження VSA зменшує витрати на апаратне забезпечення та спрощує адміністрування безпеки мережі. Python надає гнучкість для інтеграції нових механізмів аналізу загроз та адаптації до змін у кіберпросторі.

Вихідний код для системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA. Код реалізує моніторинг трафіку, аналіз загроз, фільтрацію пакетів і систему сповіщень.

```
import scapy.all as scapy
```

```
import socket
import threading
import time
import logging
import tensorflow as tf
import numpy as np
from sklearn.preprocessing import StandardScaler

# Налаштування логування
logging.basicConfig(filename="security_gateway.log", level=logging.INFO,
                    format="%(asctime)s - %(levelname)s - %(message)s")

# Функція для зчитування трафіку
def packet_sniffer(interface="eth0"):
    sniff_filter = "ip"
scapy.sniff(iface=interface, filter=sniff_filter,
            prn=analyze_packet, store=False)

# Функція аналізу пакетів
def analyze_packet(packet):
    if packet.haslayer(scapy.IP):
        src_ip = packet[scapy.IP].src
        dst_ip = packet[scapy.IP].dst
        proto = packet[scapy.IP].proto
        packet_size = len(packet)
        timestamp = time.time()

        logging.info(f"Пакет: {src_ip} -> {dst_ip},
                     Протокол: {proto}, Розмір: {packet_size} байт")

        if detect_anomaly([packet_size, proto, timestamp]):
            logging.warning(f"Аномальна активність виявлена від {src_ip}")
            block_ip(src_ip)

# Функція блокування IP
def block_ip(ip_address):
    with open("/etc/hosts.deny", "a") as deny_file:
        deny_file.write(f"ALL: {ip_address}\n")
    logging.info(f"IP {ip_address} заблоковано")

# Завантаження моделі машинного навчання
model = tf.keras.models.load_model("anomaly_detection_model.h5")
```

```
        # Масштабатор даних для нормалізації
        scaler = StandardScaler()

        # Функція виявлення аномалій
        def detect_anomaly(data):
            data = np.array(data).reshape(1, -1)
            data = scaler.transform(data)
            prediction = model.predict(data)
            return prediction[0][0] > 0.5

        # Функція для моніторингу активності
        def traffic_monitor():
            while True:
                time.sleep(10)
                logging.info("Моніторинг трафіку активний")

        # Функція для сповіщення адміністратора
        def notify_admin(message):
            admin_email = "admin@example.com"
            logging.info(f"Надсилання сповіщення адміністратору: {message}")

        # Основний потік
        if __name__ == "__main__":
            logging.info("Запуск системи кібербезпеки")

            sniffer_thread = threading.Thread(target=packet_sniffer,
args=("eth0",))
            monitor_thread = threading.Thread(target=traffic_monitor)

            sniffer_thread.start()
            monitor_thread.start()

            sniffer_thread.join()
            monitor_thread.join()
```

Код використовує бібліотеку Scapy для зчитування мережевого трафіку. Пакети проходять аналіз на предмет аномальної активності. Виявлення загроз здійснюється за допомогою попередньо навченої моделі нейронної мережі в TensorFlow.

Функція packet_sniffer захоплює мережеві пакети, використовуючи фільтр IP, і передає їх у функцію аналізу.

Функція analyze_packet отримує дані про IP-адресу відправника, одержувача, протокол і розмір пакета. Якщо виявлена підозріла активність, система блокує IP.

Функція block_ip додає підозрілу IP-адресу до списку заблокованих у файлі /etc/hosts.deny.

Функція detect_anomaly використовує модель машинного навчання для оцінки ризику загрози.

Функція traffic_monitor періодично перевіряє стан системи та зберігає інформацію у журналі.

Функція notify_admin надсилає сповіщення адміністратору про виявлену аномальну активність.

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7624:2014 («Калина»). Одним із основних алгоритмів симетричного блокового шифрування, що використовуються в Україні, є ДСТУ 7624:2014 («Калина»), який визначає сучасний алгоритм симетричного блокового перетворення для забезпечення конфіденційності й цілісності інформації при її обробці та встановлює режими його роботи.

В алгоритмі шифрування даних «Калина» використовуються криптографічні перетворення, які відповідають сучасним вимогам до рівня криптостійкості та швидкодії.

Даний стандарт розроблено з урахуванням існуючих та потенційних загроз, подальшого інтенсивного розвитку інформаційних технологій та необхідності активного використання протягом кількох наступних десятиліть.

Стандарт блокового симетричного шифрування ДСТУ 7624:2014 визначає десять різних режимів роботи, що широко поширені відповідно до міжнародних стандартів ISO/IEC 10116:2006.

Це спрямовано на забезпечення широкого застосування ДСТУ 7624:2014, у тому числі для захисту інформації, що передається комп'ютерними мережами, прозорого шифрування жорстких дисків і змінних носіїв, електронних документів, ключових даних.

Ефективність реалізації систем, засобів та протоколів криптографічного захисту інформації в інформаційно-телекомунікаційних системах різного призначення може бути забезпечена саме наявністю такої кількості режимів роботи алгоритму.

До блокового шифру «Калина» ставляться такі вимоги: високий рівень криптографічної стійкості з достатнім запасом у разі появи нових атак протягом тривалого часу; висока швидкодія програмної реалізації на сучасних та перспективних платформах; компактність програмної та програмно-апаратної реалізації; можливість ефективної інтеграції декількох алгоритмів в одному засобі криптографічного захисту; прозорість проектування, консервативний підхід до забезпечення стійкості; вища (або однакова) ефективність порівняно з найкращими світовими рішеннями.

Криптографічні алгоритми, які визначаються стандартами ДСТУ 7624:2014 і ДСТУ 7564:2014, є гнучкими, підтримують розмір блоку і довжину ключа від 128 до 512 біт.

Стандарт симетричного блокового шифрування «Калина» є результатом багаторічної плідної співпраці Державної служби спеціального зв'язку та захисту інформації України та провідних українських вчених.

Даний алгоритм шифрування враховує досвід і результати проведення міжнародних та відкритих національних конкурсів криптографічних алгоритмів.

Алгоритм ДСТУ 7624:2014 забезпечує досить високий рівень криптостікості порівняно з міжнародним стандартом AES (ISO/IEC 18033-3:2010), оскільки дає можливість застосовувати блок даних і ключ шифрування розміром аж до 512 біт. Крім того, він має аналогічну або навіть більш високу швидкодію на сучасних і перспективних програмних та програмно-апаратних

платформах. На даний момент продовжуються роботи зі стандартизації вітчизняних криптографічних алгоритмів та протоколів.

При цьому не обмежується застосування гармонізованих стандартів у сфері захисту конфіденційної інформації.

Також зусилля зосереджено на використанні кращих практик застосування стандартів шифрування даних для захисту інформації в інформаційно-комунікаційних системах [10, 11].

Оскільки в стандартах симетричного блокового шифрування «Калина» та AES використовуються аналогічні криптографічні перетворення, на наш погляд, буде доцільним порівняти ці два алгоритми.

Основними відмінностями «Калина» від «Rijndael» (AES) є: збільшена кількість циклів шифрування (запас стійкості); використання додавання за модулем 264 і за модулем 2 для введення ключової інформації (захист від алгебричних атак, лінійного та диференціального криптоаналізів, інтерполяційної атаки тощо); використання чотирьох блоків нелінійного перетворення (S-блоків) замість одного (додатковий захист від алгебричних атак, поліпшення властивостей розсіювання алгоритму – покращені статистичні властивості, відповідно, більш високий рівень стійкості до диференціального та лінійного криптоаналізів тощо); використання випадково сформованих чотирьох блоків, відібраних критеріями стійкості до диференціального, лінійного криптоаналізів, ступені нелінійності булевих функцій (на відміну від S-блоку Rijndael/Camellia та інших шифрів, що використовують звернення в полі та, відповідно, квадратичні залежності між входом і виходом, – захист від алгебричних атак); принципово нова схема створення підключів (захист від усіх відомих атак на схеми створення підключів); досить висока продуктивність; можливість відновлення сеансового ключа за окремим підключем (додатковий захист від атак, що виконують відновлення підключів).

Усі поліпшення спрямовані на збільшення стійкості та запобігання потенційним вразливостям відносно Rijndael, виявленим в останні роки [12].

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи. Розроблене програмне забезпечення шлюзів інформаційної безпеки з використанням технології VSA складається з наступних функціональних блоків:

– Навігаційне меню: Налаштування потокового шифрування; Встановлення коду доступу; Довідка.

– Функції представлені у графічному вигляді (іконки).

– Вікно виведення результату роботи системи.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Функціональних кнопок ПЗ: Сканування шлюзів; Налаштування параметрів підключення; Перегляд журналу роботи ПЗ; Запит поточного підключення.

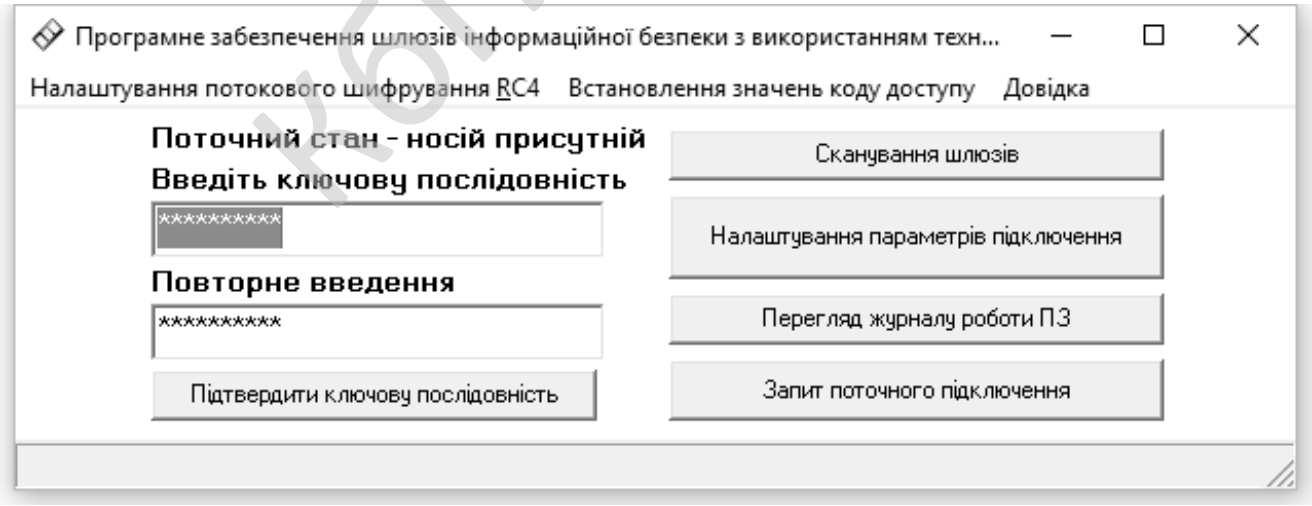

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

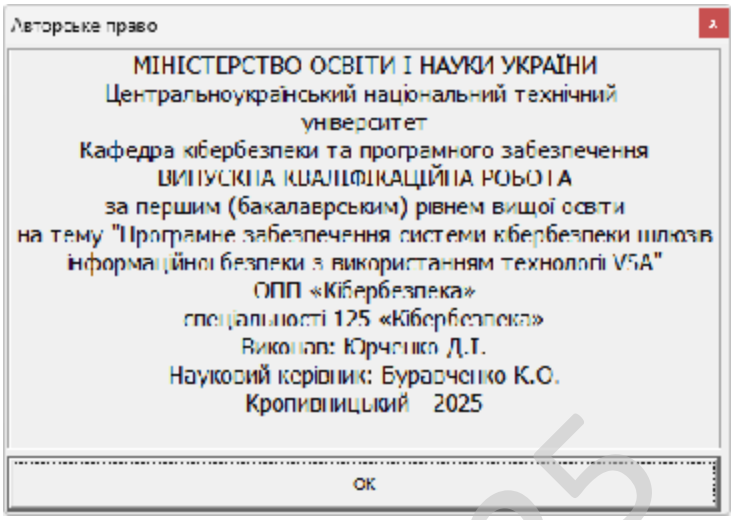

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої скриньки засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Обрано умови розповсюдження – Freeware.

Це власницьке програмне забезпечення, котре можна Безоплатно використовувати протягом необмеженого терміну без обмежень у функціональності, і поширюване без сирцевих кодів.

Автори такого програмного забезпечення, як правило, хочуть «дати щось спільноті», але хочуть також контролювати його подальшу розробку. Іноді, коли програмісти вирішують припинити розробку, вони передають сирцевий код іншим програмістам, або ж спільності як вільне програмне забезпечення.

Дуже часто плутають поняття «безплатне програмне забезпечення» та «вільне програмне забезпечення», хоча вони суттєво відрізняються.

Безплатне програмне забезпечення можна безоплатно встановлювати та використовувати (іноді з певними обмеженнями, як, наприклад, «безплатне для домашнього або некомерційного вжитку»), в той час як вільне програмне забезпечення можна продавати за будь-яку суму, але при тому, у користувача, котрий його отримує, повинні бути права на вивчення, модифікацію та поширення сирцевих кодів одержаної програми.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем шлюзів інформаційної безпеки з використанням технології VSA.

– Досліджена система шлюзів інформаційної безпеки з використанням технології VSA.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання шлюзів інформаційної безпеки з використанням технології VSA.

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 кібербезпеки шлюзів інформаційної безпеки з використанням технології

VSA. 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 кібербезпеки 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 ДСТУ 7624:2014.

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

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

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

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

5. 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.

6. 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.

7. 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.

8. 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.

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

10. 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

11. 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.

12. 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.

13. 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,

14. 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.

15. 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.

16. 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

17. 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.

18. 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.

19. 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.*

20. 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.

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

22. 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.

23. 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.

24. 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.

25. 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.

26. 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.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. 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.

39. 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.

40. 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.

41. 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.

42. 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.

43. 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.

44. 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.

45. 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.

46. 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.

47. 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.

48. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

49. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

50. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

51. Смірнова Т.В., Гнатюк С.О., Бердибаєв Р.Ш., Сидоренко В.М., Жигаревич О.К., «Система корелювання подій та управління інцидентами кібербезпеки на об’єктах критичної інфраструктури». *Кібербезпека: освіта, наука, техніка*, №*3*(19), 2023, С. 176-196.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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. № 57-02 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 кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA.

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 кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA;

– 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 – 129 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 4.06.2025 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

_____ Буравченко К.О.

Програмне забезпечення системи кібербезпеки шлюзів інформаційної безпеки з використанням технології VSA

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 20

Літера: РП

Кропивницький – 2025 року

Основна програма

```
#!/usr/bin/env python3
import requests
import threading
import time
import random
import logging
import queue
import hashlib
import json

# Налаштування базового логування системи
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s -
%(message)s')

# Глобальна константа для API URL VSA
API_URL = "http://localhost:8000/api"

# Клас VSAEngine для аналізу трафіку та роботи з VSA сервером
class VSAEngine:
    def __init__(self, api_url):
        # Ініціалізація VSAEngine з API URL
        self.api_url = api_url
        # Ініціалізація внутрішнього сховища сигнатур загроз
        self.threat_signatures = {}
        # Ініціалізація HTTP сесії для запитів
        self.session = requests.Session()
        # Логування створення об'єкта VSAEngine
        logging.debug("VSAEngine ініціалізовано з API URL: " + self.api_url)

    def update_signature(self, signature_id, signature_data):
        # Оновлення сигнатури загрози з унікальним ідентифікатором
        self.threat_signatures[signature_id] = signature_data
        # Логування оновлення сигнатури
        logging.debug("Оновлено сигнатуру: " + signature_id)

    def analyze_traffic(self, traffic_data):
        # Аналіз вхідних даних трафіку з використанням локальних сигнатур
        score = 0
        # Прохід по кожній сигнатурі загрози
        for sig_id, sig_data in self.threat_signatures.items():
            # Перевірка наявності сигнатури у даних трафіку
            if sig_data in traffic_data:
                # Збільшення рейтингу загрози при виявленні сигнатури
                score += 10
                # Логування виявлення сигнатури
                logging.debug("Виявлено сигнатуру " + sig_id + " у трафіку")
        # Повернення розрахованого рейтингу загрози
        return score

    def send_query(self, traffic_data):
        # Відправлення даних трафіку на сервер VSA для додаткового аналізу
        try:
            # Виконання HTTP POST запиту з даними трафіку у форматі JSON
            response = self.session.post(self.api_url, json={'data':
traffic_data}, timeout=5)
            # Логування успішного запиту до VSA сервера
            logging.debug("Запит до VSA сервера виконано успішно")
            # Перевірка статусу відповіді
            if response.status_code == 200:
                # Повернення відповіді сервера у форматі JSON
                return response.json()
```

```
            else:
                # Логування невдалого запиту з відповідним статусом
                logging.error("Не вдалося отримати відповідь від VSA сервера,
статус: " + str(response.status_code))
                return None
        except Exception as e:
            # Обробка виключень під час запиту до VSA сервера
            logging.error("Помилка під час запиту до VSA сервера: " + str(e))
            return None

    def refresh_signatures(self):
        # Оновлення сигнатур загроз з віддаленого сервера
        try:
            # Виконання HTTP GET запиту для отримання нових сигнатур
            response = self.session.get(self.api_url + "/signatures", timeout=5)
            # Логування спроби оновлення сигнатур
            logging.debug("Спроба оновлення сигнатур віддаленого сервера")
            if response.status_code == 200:
                # Завантаження сигнатур з відповіді
                signatures = response.json()
                # Оновлення локального словника сигнатур
                for sig in signatures:
                    # Оновлення кожної сигнатури
                    self.threat_signatures[sig['id']] = sig['pattern']
                    # Логування оновлення кожної сигнатури
                    logging.debug("Сигнатура " + sig['id'] + " оновлена")
            else:
                # Логування невдалої спроби оновлення сигнатур
                logging.error("Не вдалося оновити сигнатури з віддаленого
сервера")
        except Exception as e:
            # Обробка виключень під час оновлення сигнатур
            logging.error("Виняткова ситуація при оновленні сигнатур: " +
str(e))

    def log_internal_state(self):
        # Логування поточного стану VSAEngine
        logging.debug("Поточний стан VSAEngine:")
        # Логування API URL
        logging.debug("API URL: " + self.api_url)
        # Логування кількості сигнатур загроз
        logging.debug("Кількість сигнатур: " + str(len(self.threat_signatures)))

# Клас ThreatAnalyzer для аналізу загроз у даних трафіку
class ThreatAnalyzer:
    def __init__(self):
        # Ініціалізація ThreatAnalyzer з базовими налаштуваннями
        self.threat_database = {}
        # Завантаження стандартних сигнатур загроз
        self.load_default_signatures()
        # Логування створення ThreatAnalyzer
        logging.debug("ThreatAnalyzer ініціалізовано та стандартні сигнатури
завантажено")

    def load_default_signatures(self):
        # Завантаження стандартних сигнатур загроз у базу даних
        self.threat_database = {
            "T001": "malicious_pattern_1",
            "T002": "malicious_pattern_2",
            "T003": "suspicious_pattern_alpha",
            "T004": "suspicious_pattern_beta"
        }
        # Логування завантаження стандартних сигнатур
```

```
        logging.debug("Стандартні сигнатури загроз завантажено")

    def analyze_data(self, traffic_data):
        # Аналіз даних трафіку з використанням бази даних загроз
        threat_score = 0
        # Прохід по кожній сигнатурі в базі даних
        for threat_id, pattern in self.threat_database.items():
            # Перевірка наявності патерну загрози у даних трафіку
            if pattern in traffic_data:
                # Збільшення рейтингу загрози при виявленні патерну
                threat_score += 15
                # Логування виявлення патерну загрози
                logging.debug("Виявлено патерн загрози " + threat_id + " у даних
трафіку")
        # Повернення розрахованого рейтингу загрози
        return threat_score

    def get_threat_level(self, score):
        # Визначення рівня загрози на основі рейтингу
        if score >= 50:
            # Високий рівень загрози
            return "Високий"
        elif score >= 20:
            # Середній рівень загрози
            return "Середній"
        else:
            # Низький рівень загрози
            return "Низький"

    def update_threat_database(self, new_signatures):
        # Оновлення бази даних загроз новими сигнатурами
        for threat_id, pattern in new_signatures.items():
            # Додавання або оновлення запису у базі даних
            self.threat_database[threat_id] = pattern
            # Логування оновлення бази даних загроз
            logging.debug("База даних загроз оновлена: " + threat_id)

    def print_threat_database(self):
        # Вивід поточного стану бази даних загроз у консоль
        logging.debug("Поточний стан бази даних загроз:")
        for threat_id, pattern in self.threat_database.items():
            # Вивід кожного запису загрози
            logging.debug(threat_id + ": " + pattern)

# Клас AlertManager для відправки та логування сповіщень про загрози
class AlertManager:
    def __init__(self):
        # Ініціалізація AlertManager з налаштуваннями сповіщень
        self.alert_recipients = []
        # Завантаження стандартного списку отримувачів сповіщень
        self.load_default_recipients()
        # Логування ініціалізації AlertManager
        logging.debug("AlertManager ініціалізовано з отримувачами сповіщень")

    def load_default_recipients(self):
        # Завантаження стандартного списку отримувачів сповіщень
        self.alert_recipients = ["admin@example.com", "security@example.com"]
        # Логування завантаження отримувачів сповіщень
        logging.debug("Отримувачі сповіщень завантажені")

    def send_alert(self, alert_message):
        # Відправлення сповіщення всім отримувачам
        for recipient in self.alert_recipients:
```

```
            # Симуляція відправлення сповіщення
            logging.info("Сповіщення надіслано до " + recipient + ": " +
alert_message)

    def log_alert(self, alert_message):
        # Логування сповіщення в системний журнал
        logging.warning("Системне сповіщення: " + alert_message)

    def archive_alert(self, alert_message):
        # Архівація сповіщення для подальшого аналізу
        try:
            # Запис сповіщення у файл архіву
            with open("alert_archive.log", "a") as archive_file:
                archive_file.write(time.asctime() + " - " + alert_message +
"\n")
            # Логування успішної архівації сповіщення
            logging.debug("Сповіщення успішно архівовано")
        except Exception as e:
            # Логування помилки архівації сповіщення
            logging.error("Помилка архівації сповіщення: " + str(e))

# Клас SecurityGateway для інтеграції VSAEngine, ThreatAnalyzer та AlertManager
class SecurityGateway:
    def __init__(self, vsa_engine, threat_analyzer, alert_manager):
        # Ініціалізація шлюзу безпеки з компонентами VSAEngine, ThreatAnalyzer
та AlertManager
        self.vsa_engine = vsa_engine
        self.threat_analyzer = threat_analyzer
        self.alert_manager = alert_manager
        # Встановлення порогу загрози для сповіщень
        self.threat_threshold = 40
        # Логування створення SecurityGateway
        logging.debug("SecurityGateway ініціалізовано з порогом загрози " +
str(self.threat_threshold))

    def process_traffic(self, traffic_data):
        # Обробка даних трафіку через VSAEngine для попереднього аналізу
        vsa_score = self.vsa_engine.analyze_traffic(traffic_data)
        # Логування отриманого рейтингу з VSAEngine
        logging.debug("Рейтинг загрози від VSAEngine: " + str(vsa_score))
        # Відправка запиту до VSA сервера для додаткової перевірки
        vsa_response = self.vsa_engine.send_query(traffic_data)
        # Обробка відповіді від VSA сервера
        if vsa_response is not None and 'score' in vsa_response:
            # Додавання зовнішнього рейтингу до внутрішнього
            vsa_score += vsa_response['score']
            # Логування додаткового рейтингу загрози від VSA сервера
            logging.debug("Додатковий рейтинг від VSA сервера: " +
str(vsa_response['score']))
        # Аналіз даних трафіку за допомогою ThreatAnalyzer
        threat_score = self.threat_analyzer.analyze_data(traffic_data)
        # Логування рейтингу загрози від ThreatAnalyzer
        logging.debug("Рейтинг загрози від ThreatAnalyzer: " +
str(threat_score))
        # Обчислення загального рейтингу загрози
        total_score = vsa_score + threat_score
        # Логування загального рейтингу загрози
        logging.debug("Загальний рейтинг загрози: " + str(total_score))
        # Визначення рівня загрози
        threat_level = self.threat_analyzer.get_threat_level(total_score)
        # Логування визначеного рівня загрози
        logging.debug("Визначений рівень загрози: " + threat_level)
        # Якщо загальний рейтинг перевищує поріг, генерується сповіщення
```

```
        if total_score >= self.threat_threshold:
            # Формування повідомлення сповіщення
            alert_message = "Виявлено загрозу рівня " + threat_level + " з
рейтингом " + str(total_score)
            # Відправлення сповіщення через AlertManager
            self.alert_manager.send_alert(alert_message)
            # Логування сповіщення
            self.alert_manager.log_alert(alert_message)
            # Архівація сповіщення
            self.alert_manager.archive_alert(alert_message)
        # Повернення загального рейтингу загрози для подальшого аналізу
        return total_score

    def simulate_traffic_handling(self, traffic_queue):
        # Симуляція обробки трафіку з використанням черги запитів
        while True:
            # Спроба отримання даних з черги
            try:
                traffic_data = traffic_queue.get(timeout=1)
                # Логування отримання даних з черги
                logging.debug("Отримано дані трафіку з черги")
                # Обробка отриманих даних трафіку
                self.process_traffic(traffic_data)
                # Позначення завдання як виконаного
                traffic_queue.task_done()
            except queue.Empty:
                # Логування відсутності даних у черзі
                logging.debug("Черга трафіку порожня, очікування нових даних")
                # Затримка перед наступною спробою отримання даних
                time.sleep(0.5)

    def reset_gateway(self):
        # Скидання стану шлюзу безпеки
        logging.debug("Скидання стану шлюзу безпеки")
        # Оновлення сигнатур VSAEngine
        self.vsa_engine.refresh_signatures()
        # Перезавантаження бази даних загроз ThreatAnalyzer
        self.threat_analyzer.load_default_signatures()

# Клас TrafficSimulator для генерації випадкових даних трафіку
class TrafficSimulator:
    def __init__(self, traffic_queue):
        # Ініціалізація TrafficSimulator з чергою для даних трафіку
        self.traffic_queue = traffic_queue
        # Встановлення параметрів генерації випадкових даних трафіку
        self.patterns = ["normal_traffic", "malicious_pattern_1",
"suspicious_pattern_alpha", "benign_traffic"]
        # Логування створення TrafficSimulator
        logging.debug("TrafficSimulator ініціалізовано з " +
str(len(self.patterns)) + " паттернами")

    def generate_random_traffic(self):
        # Генерація випадкових даних трафіку
        traffic_data = ""
        # Випадкова кількість сегментів даних
        segments = random.randint(5, 15)
        # Генерація кожного сегменту даних трафіку
        for i in range(segments):
            # Вибір випадкового патерну трафіку
            pattern = random.choice(self.patterns)
            # Додавання патерну до даних трафіку
            traffic_data += pattern + " "
            # Логування генерації сегменту трафіку
```

```
            logging.debug("Згенеровано сегмент трафіку: " + pattern)
        # Повернення сформованих даних трафіку
        return traffic_data.strip()

    def simulate(self):
        # Безкінечна симуляція генерації даних трафіку
        while True:
            # Генерація нових даних трафіку
            traffic_data = self.generate_random_traffic()
            # Додавання даних трафіку до черги
            self.traffic_queue.put(traffic_data)
            # Логування додавання даних до черги
            logging.debug("Нові дані трафіку додано до черги")
            # Затримка між генерацією даних трафіку
            time.sleep(random.uniform(0.5, 2.0))

# Допоміжна функція для обробки даних
def auxiliary_processing(data):
    # Початок допоміжної обробки даних
    processed_data = data.lower()
    # Перетворення даних у нижній регістр
    processed_data = processed_data.replace(" ", "_")
    # Замінювання пробілів на підкреслення
    processed_data = hashlib.sha256(processed_data.encode()).hexdigest()
    # Обчислення хеш-значення даних
    time.sleep(0.1)
    # Затримка для симуляції обробки
    return processed_data

# Функція для виконання рутинної перевірки системи
def perform_routine_check():
    # Початок рутинної перевірки системи
    logging.debug("Виконання рутинної перевірки системи")
    time.sleep(0.2)
    # Завершення рутинної перевірки системи
    logging.debug("Рутинна перевірка системи завершена")

# Функція для моніторингу стану системи
def system_health_monitor():
    # Початок моніторингу стану системи
    for i in range(3):
        logging.debug("Моніторинг стану системи, цикл " + str(i+1))
        time.sleep(0.1)
    # Завершення моніторингу стану системи
    logging.debug("Моніторинг стану системи завершено")

# Функція для періодичного оновлення сигнатур в VSAEngine
def periodic_signature_update(vsa_engine):
    # Початок періодичного оновлення сигнатур
    while True:
        vsa_engine.refresh_signatures()
        logging.debug("Періодичне оновлення сигнатур завершено")
        time.sleep(10)

# Функція для перевірки системи сповіщень AlertManager
def alert_system_check(alert_manager):
    # Початок перевірки системи сповіщень
    logging.debug("Виконується перевірка системи сповіщень")
    time.sleep(0.3)
    # Завершення перевірки системи сповіщень
    logging.debug("Перевірка системи сповіщень завершена")

# Допоміжна рутина для збільшення обсягу коду (рутина один)
```

```
def dummy_routine_one():
    # Початок виконання допоміжної рутини один
    logging.debug("Допоміжна рутина один розпочата")
    for i in range(5):
        logging.debug("Допоміжна рутина один, ітерація " + str(i+1))
        time.sleep(0.05)
    logging.debug("Допоміжна рутина один завершена")

# Допоміжна рутина для збільшення обсягу коду (рутина два)
def dummy_routine_two():
    # Початок виконання допоміжної рутини два
    logging.debug("Допоміжна рутина два розпочата")
    for i in range(3):
        logging.debug("Допоміжна рутина два, ітерація " + str(i+1))
        time.sleep(0.07)
    logging.debug("Допоміжна рутина два завершена")

# Функція для симуляції запитів до VSAEngine
def simulate_queries(vsa_engine, traffic_data):
    # Початок симуляції запитів до VSAEngine
    for i in range(4):
        logging.debug("Симуляція запиту " + str(i+1) + " до VSAEngine")
        response = vsa_engine.send_query(traffic_data)
        if response is not None:
            logging.debug("Отримано відповідь для запиту " + str(i+1) + ": " +
str(response))
        else:
            logging.debug("Запит " + str(i+1) + " не дав відповіді")
        time.sleep(0.2)
    logging.debug("Симуляція запитів до VSAEngine завершена")

# Додаткова допоміжна функція для збільшення обсягу коду
def extra_dummy_function():
    # Початок виконання додаткової допоміжної функції
    for i in range(10):
        logging.debug("Додаткова допоміжна функція, ітерація " + str(i+1))
        result = i * i
        logging.debug("Результат обчислення: " + str(result))
        time.sleep(0.03)
    logging.debug("Додаткова допоміжна функція завершена")

# Функція для формування звіту про стан системи
def system_status_report():
    # Початок формування звіту про стан системи
    report = {
        "time": time.asctime(),
        "status": "Operational",
        "active_threads": threading.active_count()
    }
    logging.debug("Звіт про стан системи: " + json.dumps(report))
    return report

# Функція для симуляції навантаження на систему
def simulate_load():
    # Початок симуляції навантаження на систему
    for i in range(5):
        delay = random.uniform(0.1, 0.5)
        logging.debug("Симуляція навантаження, затримка " + str(round(delay, 2))
+ " секунд")
        time.sleep(delay)
    logging.debug("Симуляція навантаження завершена")

# Комплексна перевірка системи, що викликає декілька допоміжних функцій
```

```
def comprehensive_system_check():
    # Початок комплексної перевірки системи
    logging.debug("Комплексна перевірка системи розпочата")
    perform_routine_check()
    system_health_monitor()
    extra_dummy_function()
    simulate_load()
    system_status_report()
    logging.debug("Комплексна перевірка системи завершена")

# Функція для фіналізації роботи системи
def finalize_system():
    # Початок фіналізації роботи системи
    logging.debug("Фіналізація роботи системи розпочата")
    time.sleep(0.2)
    logging.debug("Фіналізація роботи системи завершена")

# Головна функція програми
def main():
    # Початок головного виконання програми
    api_url = API_URL
    vsa_engine = VSAEngine(api_url)
    threat_analyzer = ThreatAnalyzer()
    alert_manager = AlertManager()
    security_gateway = SecurityGateway(vsa_engine, threat_analyzer,
alert_manager)
    traffic_queue = queue.Queue()
    traffic_simulator = TrafficSimulator(traffic_queue)
    traffic_thread = threading.Thread(target=traffic_simulator.simulate,
daemon=True)
    traffic_thread.start()
    gateway_thread =
threading.Thread(target=security_gateway.simulate_traffic_handling,
args=(traffic_queue,), daemon=True)
    gateway_thread.start()
    signature_thread = threading.Thread(target=periodic_signature_update,
args=(vsa_engine,), daemon=True)
    signature_thread.start()
    try:
        while True:
            perform_routine_check()
            system_health_monitor()
            dummy_routine_one()
            dummy_routine_two()
            sample_traffic = "normal_traffic malicious_pattern_1
suspicious_pattern_alpha"
            simulate_queries(vsa_engine, sample_traffic)
            comprehensive_system_check()
            system_status_report()
            finalize_system()
            time.sleep(5)
    except KeyboardInterrupt:
        logging.info("Програма припиняє роботу через KeyboardInterrupt")
    logging.info("Головний процес завершено")

if __name__ == '__main__':
    # Запуск головної функції програми
    main()
```

Файл LogBlockchain.py

```
import tkinter as tk
from tkinter import ttk
import sqlite3
import datetime
import threading
import hashlib
import json
import random
import time
from flask import Flask, request, jsonify
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

class TrafficMLAnalyzer:
    def __init__(self):
        self.model = LogisticRegression()
        self.trained = False
        self.data = None
        self.labels = None
    def generate_synthetic_data(self, n_samples=1000):
        X = np.random.rand(n_samples, 10)
        y = (np.sum(X, axis=1) > 5).astype(int)
        self.data = X
        self.labels = y
    def train_model(self):
        if self.data is None or self.labels is None:
            self.generate_synthetic_data()
        X_train, X_test, y_train, y_test = train_test_split(self.data,
self.labels, test_size=0.2)
        self.model.fit(X_train, y_train)
        predictions = self.model.predict(X_test)
        acc = accuracy_score(y_test, predictions)
        self.trained = True
        return acc
    def predict_traffic(self, traffic_features):
        if not self.trained:
            self.train_model()
        prediction = self.model.predict([traffic_features])
        return prediction[0]

class ThreatHistoryDB:
    def __init__(self, db_name='threat_history.db'):
        self.connection = sqlite3.connect(db_name, check_same_thread=False)
        self.cursor = self.connection.cursor()
        self.create_table()
    def create_table(self):
        self.cursor.execute("CREATE TABLE IF NOT EXISTS threats (id INTEGER
PRIMARY KEY AUTOINCREMENT, timestamp TEXT, threat_level TEXT, description
TEXT)")
        self.connection.commit()
    def insert_threat(self, threat_level, description):
        timestamp = datetime.datetime.now().isoformat()
        self.cursor.execute("INSERT INTO threats (timestamp, threat_level,
description) VALUES (?, ?, ?)", (timestamp, threat_level, description))
        self.connection.commit()
    def update_threat(self, threat_id, threat_level, description):
        self.cursor.execute("UPDATE threats SET threat_level = ?, description =
? WHERE id = ?", (threat_level, description, threat_id))
        self.connection.commit()
```

```
    def delete_threat(self, threat_id):
        self.cursor.execute("DELETE FROM threats WHERE id = ?", (threat_id,))
        self.connection.commit()
    def fetch_all_threats(self):
        self.cursor.execute("SELECT * FROM threats")
        return self.cursor.fetchall()
    def fetch_threat_by_id(self, threat_id):
        self.cursor.execute("SELECT * FROM threats WHERE id = ?", (threat_id,))
        return self.cursor.fetchone()
    def close(self):
        self.connection.close()

class LogBlockchain:
    def __init__(self):
        self.chain = []
        self.create_genesis_block()
    def create_genesis_block(self):
        genesis_block = {'index': 0, 'timestamp': str(datetime.datetime.now()),
'data': 'Genesis Block', 'previous_hash': '0'}
        genesis_block['hash'] = self.compute_hash(genesis_block)
        self.chain.append(genesis_block)
    def compute_hash(self, block):
        block_string = json.dumps(block, sort_keys=True)
        return hashlib.sha256(block_string.encode()).hexdigest()
    def add_block(self, data):
        previous_block = self.chain[-1]
        new_block = {'index': previous_block['index'] + 1, 'timestamp':
str(datetime.datetime.now()), 'data': data, 'previous_hash':
previous_block['hash']}
        new_block['hash'] = self.compute_hash(new_block)
        self.chain.append(new_block)
        return new_block
    def is_chain_valid(self):
        for i in range(1, len(self.chain)):
            current = self.chain[i]
            previous = self.chain[i-1]
            if current['previous_hash'] != previous['hash']:
                return False
            if current['hash'] != self.compute_hash(current):
                return False
        return True
    def get_chain(self):
        return self.chain

class SecurityGUI:
    def __init__(self, root, threat_db, ml_analyzer, blockchain):
        self.root = root
        self.threat_db = threat_db
        self.ml_analyzer = ml_analyzer
        self.blockchain = blockchain
        self.root.title("Security Dashboard")
        self.setup_ui()
    def setup_ui(self):
        self.frame_top = ttk.Frame(self.root)
        self.frame_top.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
        self.frame_bottom = ttk.Frame(self.root)
        self.frame_bottom.pack(side=tk.BOTTOM, fill=tk.X)
        self.btn_refresh = ttk.Button(self.frame_bottom, text="Refresh",
command=self.refresh_data)
        self.btn_refresh.pack(side=tk.LEFT, padx=5, pady=5)
        self.tree = ttk.Treeview(self.frame_top, columns=("ID", "Timestamp",
"Threat Level", "Description"), show='headings')
        self.tree.heading("ID", text="ID")
```

```
        self.tree.heading("Timestamp", text="Timestamp")
        self.tree.heading("Threat Level", text="Threat Level")
        self.tree.heading("Description", text="Description")
        self.tree.pack(fill=tk.BOTH, expand=True)
        self.text_blockchain = tk.Text(self.root, height=10)
        self.text_blockchain.pack(fill=tk.BOTH, expand=True)
    def refresh_data(self):
        for item in self.tree.get_children():
            self.tree.delete(item)
        threats = self.threat_db.fetch_all_threats()
        for threat in threats:
            self.tree.insert("", tk.END, values=threat)
        self.text_blockchain.delete(1.0, tk.END)
        chain = self.blockchain.get_chain()
        for block in chain:
            self.text_blockchain.insert(tk.END, json.dumps(block) + "\n")

def run_flask_api(threat_db, ml_analyzer, blockchain):
    app = Flask(__name__)
    @app.route('/api/threats', methods=['GET'])
    def get_threats():
        threats = threat_db.fetch_all_threats()
        return jsonify(threats)
    @app.route('/api/threats/<int:threat_id>', methods=['GET'])
    def get_threat(threat_id):
        threat = threat_db.fetch_threat_by_id(threat_id)
        return jsonify(threat)
    @app.route('/api/threats', methods=['POST'])
    def add_threat():
        data = request.get_json()
        threat_level = data.get('threat_level', 'Unknown')
        description = data.get('description', '')
        threat_db.insert_threat(threat_level, description)
        blockchain.add_block({"threat_level": threat_level, "description":
description})
        return jsonify({"status": "success"})
    @app.route('/api/predict', methods=['POST'])
    def predict():
        data = request.get_json()
        features = data.get('features')
        prediction = ml_analyzer.predict_traffic(features)
        return jsonify({"prediction": int(prediction)})
    @app.route('/api/blockchain', methods=['GET'])
    def get_blockchain():
        chain = blockchain.get_chain()
        return jsonify(chain)
    app.run(port=5000, threaded=True)

def start_api(threat_db, ml_analyzer, blockchain):
    api_thread = threading.Thread(target=run_flask_api, args=(threat_db,
ml_analyzer, blockchain))
    api_thread.daemon = True
    api_thread.start()

def main():
    ml_analyzer = TrafficMLAnalyzer()
    ml_accuracy = ml_analyzer.train_model()
    threat_db = ThreatHistoryDB()
    blockchain = LogBlockchain()
    for i in range(5):
        level = "High" if random.random() > 0.5 else "Low"
        threat_db.insert_threat(level, "Threat detected sample " + str(i))
        blockchain.add_block({"threat_sample": i, "level": level})
```

```
    start_api(threat_db, ml_analyzer, blockchain)
    root = tk.Tk()
    gui = SecurityGUI(root, threat_db, ml_analyzer, blockchain)
    gui.refresh_data()
    root.mainloop()

if __name__ == '__main__':
    main()
```

Файл IncidentResponseManager.py

```
import threading
import time
import random
import requests
import json
import base64
import hashlib
import os
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from cryptography.fernet import Fernet

class IncidentResponseManager:
    def __init__(self):
        self.blocked_ips = set()
        self.response_log = []
    def auto_block_ip(self, ip_address):
        self.blocked_ips.add(ip_address)
        self.response_log.append({"action": "block", "ip": ip_address,
"timestamp": time.time()})
    def update_firewall_rules(self, rule):
        self.response_log.append({"action": "update_firewall", "rule": rule,
"timestamp": time.time()})
    def notify_admin(self, message):
        self.response_log.append({"action": "notify", "message": message,
"timestamp": time.time()})
    def process_incident(self, threat_level, ip_address):
        if threat_level == "High":
            self.auto_block_ip(ip_address)
            self.update_firewall_rules("Deny all from " + ip_address)
            self.notify_admin("High threat detected from " + ip_address)
        elif threat_level == "Medium":
            self.notify_admin("Medium threat detected from " + ip_address)
        else:
            self.response_log.append({"action": "log", "message": "Low threat
from " + ip_address, "timestamp": time.time()})
    def get_response_log(self):
        return self.response_log
    def get_blocked_ips(self):
        return list(self.blocked_ips)
    def reset_response_log(self):
        self.response_log = []
    def simulate_response_delay(self, delay=1):
        time.sleep(delay)
    def batch_process_incidents(self, incidents):
        for incident in incidents:
            self.process_incident(incident.get("threat_level", "Low"),
incident.get("ip", "0.0.0.0"))
            self.simulate_response_delay(0.1)

class DataProtector:
    def __init__(self, key=None):
        if key is None:
            self.key = Fernet.generate_key()
        else:
            self.key = key
        self.cipher = Fernet(self.key)
    def encrypt_data(self, data):
        if isinstance(data, str):
            data = data.encode()
        encrypted = self.cipher.encrypt(data)
```

```
        return encrypted
    def decrypt_data(self, token):
        decrypted = self.cipher.decrypt(token)
        return decrypted.decode()
    def rotate_key(self):
        new_key = Fernet.generate_key()
        self.key = new_key
        self.cipher = Fernet(new_key)
        return new_key
    def secure_hash(self, data):
        if isinstance(data, str):
            data = data.encode()
        return hashlib.sha256(data).hexdigest()
    def simulate_data_protection(self, data, iterations=3):
        result = data
        for _ in range(iterations):
            result = self.encrypt_data(result)
            result = self.decrypt_data(result)
        return result

class AntivirusScanner:
    def __init__(self):
        self.virus_signatures = {"eicar_test":
"44d88612fea8a8f36de82e1278abb02f"}
    def scan_file(self, file_path):
        if not os.path.isfile(file_path):
            return {"status": "error", "message": "File not found"}
        with open(file_path, "rb") as f:
            file_data = f.read()
        file_hash = hashlib.md5(file_data).hexdigest()
        for name, signature in self.virus_signatures.items():
            if file_hash == signature:
                return {"status": "infected", "virus": name}
        return {"status": "clean"}
    def update_signatures(self, new_signatures):
        self.virus_signatures.update(new_signatures)
        return self.virus_signatures
    def scan_directory(self, dir_path):
        results = {}
        if not os.path.isdir(dir_path):
            return {"status": "error", "message": "Directory not found"}
        for root, dirs, files in os.walk(dir_path):
            for file in files:
                file_path = os.path.join(root, file)
                results[file_path] = self.scan_file(file_path)
        return results
    def simulate_full_system_scan(self, dir_path):
        scan_results = self.scan_directory(dir_path)
        infected_files = [file for file, result in scan_results.items() if
result.get("status") == "infected"]
        return {"total_files": len(scan_results), "infected": infected_files}

class DDoSDetector:
    def __init__(self, threshold=100, interval=60):
        self.threshold = threshold
        self.interval = interval
        self.request_counts = {}
        self.lock = threading.Lock()
    def record_request(self, ip_address):
        current_time = int(time.time())
        with self.lock:
            if ip_address not in self.request_counts:
                self.request_counts[ip_address] = []
```

```
            self.request_counts[ip_address].append(current_time)
    def clean_old_requests(self):
        current_time = int(time.time())
        with self.lock:
            for ip in list(self.request_counts.keys()):
                self.request_counts[ip] = [t for t in self.request_counts[ip] if
current_time - t <= self.interval]
                if not self.request_counts[ip]:
                    del self.request_counts[ip]
    def detect_ddos(self):
        self.clean_old_requests()
        alerts = []
        with self.lock:
            for ip, times in self.request_counts.items():
                if len(times) > self.threshold:
                    alerts.append(ip)
        return alerts
    def run_detection_loop(self):
        while True:
            ddos_ips = self.detect_ddos()
            if ddos_ips:
                print("DDoS detected from IPs:", ddos_ips)
            time.sleep(self.interval)
    def get_all_request_counts(self):
        with self.lock:
            return dict(self.request_counts)
    def simulate_traffic(self, ip_address, num_requests):
        for _ in range(num_requests):
            self.record_request(ip_address)
            time.sleep(random.uniform(0.01, 0.05))

class ThreatIntelligenceFetcher:
    def __init__(self, api_url):
        self.api_url = api_url
        self.cache = {}
        self.lock = threading.Lock()
    def fetch_threat_data(self):
        try:
            response = requests.get(self.api_url, timeout=5)
            if response.status_code == 200:
                data = response.json()
                with self.lock:
                    self.cache = data
                return data
            return None
        except Exception:
            return None
    def get_cached_data(self):
        with self.lock:
            return self.cache
    def clear_cache(self):
        with self.lock:
            self.cache = {}
    def periodic_fetch(self, interval=300):
        while True:
            self.fetch_threat_data()
            time.sleep(interval)
    def simulate_threat_update(self, new_data):
        with self.lock:
            self.cache.update(new_data)
        return self.cache

def main():
```

```
    irm = IncidentResponseManager()
    dp = DataProtector()
    av = AntivirusScanner()
    ddos = DDoSDetector(threshold=50, interval=30)
    tif = ThreatIntelligenceFetcher("https://api.example.com/threats")
    threading.Thread(target=ddos.run_detection_loop, daemon=True).start()
    threading.Thread(target=tif.periodic_fetch, daemon=True).start()
    sample_ips = ["192.168.1.1", "10.0.0.1", "172.16.0.1"]
    for ip in sample_ips:
        irm.process_incident("High", ip)
    incidents = [{"threat_level": "Medium", "ip": "192.168.2.1"},
{"threat_level": "Low", "ip": "10.0.2.1"}]
    irm.batch_process_incidents(incidents)
    blocked = irm.get_blocked_ips()
    log = irm.get_response_log()
    rotated_key = dp.rotate_key()
    secured = dp.secure_hash("Important Data")
    simulated = dp.simulate_data_protection("Sensitive data", iterations=5)
    print("Encrypted:", dp.encrypt_data("Secret Info"))
    print("Decrypted:", dp.decrypt_data(dp.encrypt_data("Secret Info")))
    scan_result = av.scan_file("sample_file.txt")
    dir_scan = av.scan_directory(".")
    full_scan = av.simulate_full_system_scan(".")
    print("Antivirus scan result:", scan_result)
    print("Directory scan result:", dir_scan)
    print("Full system scan:", full_scan)
    for _ in range(60):
        ddos.record_request("192.168.1.100")
        time.sleep(0.05)
    ddos.simulate_traffic("10.0.0.200", 80)
    request_counts = ddos.get_all_request_counts()
    fetched_data = tif.fetch_threat_data()
    tif.simulate_threat_update({"new_threat": "example_threat_data"})
    cached = tif.get_cached_data()
    print("Blocked IPs:", blocked)
    print("Response Log:", log)
    print("Rotated Key:", rotated_key)
    print("Secure Hash:", secured)
    print("Simulated Protection:", simulated)
    print("DDoS Request Counts:", request_counts)
    print("Threat Intelligence Cached Data:", cached)
    time.sleep(5)

if __name__ == '__main__':
    main()
```

Файл SIEMIntegration.py

```
import time
import datetime
import threading
import random
import os
import json
import zipfile
import shutil

class SIEMIntegration:
    def __init__(self):
        self.logs = []
        self.alerts = []
    def add_log(self, source, message, severity):
        log_entry = {"source": source, "message": message, "severity": severity,
"timestamp": datetime.datetime.now().isoformat()}
        self.logs.append(log_entry)
    def collect_logs(self):
        sources = ["Firewall", "Antivirus", "System", "Application"]
        for _ in range(20):
            source = random.choice(sources)
            message = "Log message from " + source + " id " +
str(random.randint(1000, 9999))
            severity = random.choice(["Low", "Medium", "High"])
            self.add_log(source, message, severity)
            time.sleep(0.1)
    def correlate_events(self):
        correlated = []
        for log in self.logs:
            if log["severity"] == "High":
                correlated.append(log)
        return correlated
    def send_alerts(self):
        high_events = self.correlate_events()
        for event in high_events:
            alert = "ALERT: " + event["source"] + " - " + event["message"] + "
at " + event["timestamp"]
            self.alerts.append(alert)
    def generate_report(self):
        report = {"total_logs": len(self.logs), "high_events":
len(self.correlate_events()), "alerts": self.alerts}
        return report

class UserBehaviorAnalytics:
    def __init__(self):
        self.user_actions = {}
        self.risk_threshold = 50
    def record_action(self, user_id, action, details):
        entry = {"action": action, "details": details, "timestamp":
datetime.datetime.now().isoformat()}
        if user_id not in self.user_actions:
            self.user_actions[user_id] = []
        self.user_actions[user_id].append(entry)
    def analyze_behavior(self, user_id):
        actions = self.user_actions.get(user_id, [])
        risk_score = 0
        for act in actions:
            if act["action"] in ["login_failure", "unauthorized_access",
"data_exfiltration"]:
                risk_score += 20
            elif act["action"] in ["password_change", "login_success"]:
```

```
                risk_score += 5
            else:
                risk_score += 1
        return risk_score
    def generate_user_alert(self, user_id):
        score = self.analyze_behavior(user_id)
        if score >= self.risk_threshold:
            return "User " + str(user_id) + " has abnormal behavior with risk
score " + str(score)
        return None
    def simulate_user_activity(self):
        users = [101, 102, 103, 104, 105]
        actions = ["login_success", "login_failure", "file_access",
"unauthorized_access", "data_exfiltration", "password_change"]
        for _ in range(50):
            user = random.choice(users)
            action = random.choice(actions)
            details = "Detail " + str(random.randint(1, 100))
            self.record_action(user, action, details)
            time.sleep(0.05)
    def analyze_all_users(self):
        alerts = []
        for user in self.user_actions:
            alert = self.generate_user_alert(user)
            if alert:
                alerts.append(alert)
        return alerts

class BackupRecoverySystem:
    def __init__(self, backup_dir="backups"):
        self.backup_dir = backup_dir
        if not os.path.exists(self.backup_dir):
            os.makedirs(self.backup_dir)
        self.backup_index = {}
    def create_backup(self, source_dir):
        timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
        backup_file = os.path.join(self.backup_dir, "backup_" + timestamp +
".zip")
        with zipfile.ZipFile(backup_file, "w") as zipf:
            for foldername, subfolders, filenames in os.walk(source_dir):
                for filename in filenames:
                    file_path = os.path.join(foldername, filename)
                    arcname = os.path.relpath(file_path, source_dir)
                    zipf.write(file_path, arcname)
        self.backup_index[timestamp] = backup_file
        return backup_file
    def list_backups(self):
        return self.backup_index
    def restore_backup(self, timestamp, restore_dir):
        if timestamp in self.backup_index:
            backup_file = self.backup_index[timestamp]
            if os.path.exists(restore_dir):
                shutil.rmtree(restore_dir)
            os.makedirs(restore_dir)
            with zipfile.ZipFile(backup_file, "r") as zipf:
                zipf.extractall(restore_dir)
            return True
        return False
    def schedule_backup(self, source_dir, interval):
        def backup_task():
            while True:
                self.create_backup(source_dir)
                time.sleep(interval)
```

```
        t = threading.Thread(target=backup_task)
        t.daemon = True
        t.start()
    def simulate_backup_restore(self, source_dir, restore_dir):
        backup_file = self.create_backup(source_dir)
        time.sleep(1)
        timestamps = list(self.backup_index.keys())
        if timestamps:
            latest = sorted(timestamps)[-1]
            restored = self.restore_backup(latest, restore_dir)
            return restored
        return False

def main():
    siem = SIEMIntegration()
    uba = UserBehaviorAnalytics()
    brs = BackupRecoverySystem()
    siem.collect_logs()
    siem.send_alerts()
    siem_report = siem.generate_report()
    uba.simulate_user_activity()
    user_alerts = uba.analyze_all_users()
    source_dir = "sample_source"
    restore_dir = "sample_restore"
    if not os.path.exists(source_dir):
        os.makedirs(source_dir)
    for i in range(5):
        with open(os.path.join(source_dir, "file_" + str(i) + ".txt"), "w") as
f:
            f.write("Sample content " + str(i))
    brs.schedule_backup(source_dir, 5)
    time.sleep(6)
    backup_list = brs.list_backups()
    restore_status = brs.simulate_backup_restore(source_dir, restore_dir)
    print("SIEM Report:", json.dumps(siem_report, indent=2))
    print("UBA Alerts:", json.dumps(user_alerts, indent=2))
    print("Backups:", json.dumps(backup_list, indent=2))
    print("Restore Status:", restore_status)

if __name__ == '__main__':
    main()
```