МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Мицак Ілля Миколайович

Програмне забезпечення системи кібербезпеки для моделювання ботнет мережі

Спеціальність: 125 «Кібербезпека»

Освітній ступінь: бакалавр

Науковий керівник:

Смірнов Сергій Анатолійович _____

(підпис) (дата)

кандидат технічних наук

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *125 Кібербезпека*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Мицаку Іллі Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для моделювання ботнет мережі*

керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 185-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи кібербезпеки для моделювання ботнет мережі*

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

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

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

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

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Мицак І.М. Програмне забезпечення системи кібербезпеки для моделювання ботнет мережі. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи кібербезпеки для моделювання ботнет мережі.

Метою розробки є програмне забезпечення системи кібербезпеки для моделювання ботнет мережі.

Результат роботи – програмна реалізація системи кібербезпеки для моделювання ботнет мережі.

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 архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі Delphi 10.4 Sydney.

Ключові слова: кібербезпека, ботнет

ABSTRACT

Mytsak I.M. Cybersecurity system software for modeling a botnet network. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification the software which is intended for system of cybersecurity for modeling of a botnet network is developed.

The purpose of the development is cybersecurity system software for modeling a botnet network.

The result is a software implementation of a cybersecurity system for modeling a botnet network.

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.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the environment of Delphi 10.4 Sydney.

Keywords: cybersecurity, botnet

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ПЗ	Програмне забезпечення
DoS	Denial-of-service attack, напад на комп'ютерну систему з наміром зробити комп'ютерні ресурси недоступними користувачам, для яких комп'ютерна система була призначена.
DDoS	DoS атака, яка відбувається одночасно з великої кількості IP-адрес
DNS	Domain Name System, ієрархічна розподілена система перетворення імені хоста (комп'ютера або іншого мережевого пристрою) в IP-адресу.
IP	Internet Protocol, протокол мережевого рівня для передавання датаграм між мережами
TCP	Transmission Control Protocol, протокол призначений для управління передачею даних у комп'ютерних мережах, працює на транспортному рівні моделі OSI.
UDP	User Datagram Protocol, один з найпростіших протоколів транспортного рівня моделі OSI, котрий виконує обмін повідомленнями (датаграмами) без підтвердження та гарантії доставки

ВСТУП

Актуальність теми. Атаки ботнету Mirai на американського DNS-провайдера Dyn в 2016 році викликали широкий резонанс і привернули підвищену увагу до ботнетам. Однак, по порівнянню з тим, як сучасні кіберзлочинці використовують ботнети сьогодні, атаки на компанію Dyn можуть здатися дитячими витівками. Злочинці швидко навчилися використовувати ботнети для запуску складних шкідливих програм, що дозволяють створювати цілі інфраструктури із заражених комп'ютерів і інших пристроїв з виходом в Інтернет для одержання незаконного прибутку у величезних масштабах.

В останні роки правоохоронні органі добилися певних успіхів у боротьбі зі злочинною діяльністю, пов'язаної з використанням ботнетів, але поки цих зусиль, звичайно ж, недостатньо, щоб пробити достатній пролом у ботнетах під керуванням кіберзлочинців. От кілька відомих прикладів:

– Міністерство юстиції США висунуло проти обвинувачення двом молодим людям за їхню роль у розробці й використанні ботнету Mirai: 21-літньому Парасу Джа (Paras Jha) і 20-літньому Джошуа Вайту (Josiah White). Їх обвинувачують в організації й проведенні DDoS-атак на компанії, а потім вимозі викупу за їхнє припинення, а також із продажу цим компаніям «послуг» по запобіганню подібних атак у майбутньому.

– Іспанські влади в рамках транскордонної операції по запиту США заарештували жителя Санкт-петербурга Петра Левашова, відомого в кіберзлочинних колах як Peter Severa. Він управляв Kelihos, одним із самих довго існуючих в Інтернеті ботнетів, який, як оцінюється, заразив близько 100 тис. комп'ютерів. Крім вимагання, Петро Левашов активно використовував Kelihos для організації спам-розсилань, беручи по $200-$500 за мільйон повідомлень.

– Торік два ізраїльські тінейджера були арештовані за обвинуваченням в організації DDoS-атак за винагороду. Пара встигнула заробити близько $600 тис. і провести порядку 150 тис. DDoS-атак.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для моделювання ботнет мережі.

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

– Огляд існуючих систем кібербезпеки для моделювання ботнет мережі.

– Дослідження системи кібербезпеки для моделювання ботнет мережі.

– Програмна реалізація системи кібербезпеки для моделювання ботнет мережі.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для моделювання ботнет мережі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для моделювання ботнет мережі, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Ботнет представляє із себе мережу комп'ютерів, які інфіковані шкідливим ПЗ зловмисникам, що дозволяють, здійснювати віддалений контроль над ними, розсилати спам-повідомлення й віруси, служити місцем розміщення ПЗ, що здійснює DDoS-атаки – і все це без відома справжніх хазяїв комп'ютерів.

Для того, щоб ваш комп'ютер став частиною ботнету, він повинен бути інфікований спеціалізованим шкідливим ПЗ, яке підтримує контакт із віддаленим сервером або з іншим зараженим пристроєм, одержуючи, таким чином, інструкції до дій від зловмисників, що контролюють даний ботнет. Крім значних масштабів зараження, шкідливе ПЗ, використовуване з метою створення ботнетів, по суті мало чому відрізняється від традиційних шкідливих програм.

Виявити типове для ботнету шкідливе ПЗ можна тим же самим способом, що й у випадку з усіма іншими шкідливими програмами. Непрямими ознаками можуть бути повільна робота, дивні дії, повідомлення про помилки або раптовий запуск вентилятора охолодження під час того, як комп'ютер перебуває в режимі очікування. Це можливі симптоми того, що хтось віддалено використовує ваш комп'ютер, що став частиною розгалуженого ботнету.

Для того, щоб вилучити свій ПК із ботнету, необхідно вилучити шкідливе ПЗ, за допомогою якого зловмисники здійснюють віддалений контроль над ним. Найефективнішим способом є антивірусне сканування системи вашого комп'ютера, яке допоможе виявити шкідливу програму й відсторонити її з комп'ютера.

Як уникнути зараження шкідливим ПЗ, характерним для ботнету:

- Установіть якісний антивірусний застосунок на свій комп'ютер.
- Налаштуйте автоматичне відновлення всіх сторонніх програм.

– Будьте гранично обережні при переході по посиланнях, завантаженні програм і відкритті файлів.

Щоб убезпечити свій комп'ютер від ризику стати одним з ‘зомбі’ в армії ботнету, намагайтеся уникати будь-яких підозрілих завантажень. Не переходите по посиланнях і не відкривайте вкладених файлів з листів, відправники яких вам невідомі, і будьте гранично уважні при установці стороннього ПЗ на свій комп'ютер. Підтримуйте стороннє ПЗ оновлених і встановлюйте всі самі свіжі відновлення операційної системи. Однак найважливіше – це використання сучасного і якісного антивірусного захисту, який забезпечить надійний захист комп'ютера від усіх типів шкідливого ПЗ й допоможе уникнути інфікування вашого комп'ютера й включення його в ботнет.

1.2 Область застосування

Для реалізації програмного забезпечення системи кібербезпеки для моделювання ботнет мережі, пропонується використовувати нейронні мережі.

Нейронна мережа – спроба за допомогою математичних моделей відтворити роботу людського мозку для створення машин, що володіють штучним інтелектом.

Штучна нейронна мережа звичайно навчається із вчителем. Це означає наявність навчального набору (датасета), який містить приклади з дійсними значеннями: тегами, класами, показниками.

Нерозмічені набори також використовують для навчання нейронних мереж, але ми не будемо тут це розглядати.

Наприклад, якщо ви прагнете створити нейромережа для оцінки тональності тексту, датасетом буде список пропозицій з відповідними кожному емоційними оцінками. Тональність тексту визначають ознаки (слова, фрази, структура пропозиції), які надають негативну або позитивне фарбування. Ваги ознак у підсумковій оцінці тональності тексту (позитивний, негативний,

нейтральний) залежать від математичної функції, яка обчислюється під час навчання нейронної мережі.

Раніше люди генерували ознаки вручну. Чим більше ознак і точніше підібрані ваги, тем точніше відповідь. Нейронна мережа автоматизувала цей процес.

Штучна нейронна мережа складається із трьох компонентів:

- Вхідний шар.
- Сховані (обчислювальні) шари.
- Вихідний шар.

Навчання нейромереж відбувається у два етапи:

- Пряме поширення помилки.
- Зворотне поширення помилки.

Під час прямого поширення помилки робиться пророкування відповіді. При зворотному поширенні помилка між фактичною відповіддю й передвіщеним мінімізується.

Пряме поширення помилки

Пряме поширення

Задамо початкові ваги випадковим образом:

- w1
- w2
- w3

Помножимо вхідні дані на ваги для формування схованого шару:

$$h1 = (x1 * w1) + (x2 * w1)$$

$$h2 = (x1 * w2) + (x2 * w2)$$

$$h3 = (x1 * w3) + (x2 * w3)$$

Вихідні дані зі схованого шару передається через нелінійну функцію (функцію активації), для одержання виходу мережі:

$$y\_ = fn(h1, h2, h3)$$

Зворотне поширення

– Сумарна помилка (total_error) обчислюється як різниця між очікуваним значенням «y» (з навчального набору) і отриманим значенням «y_» (полічене на етапі прямого поширення помилки), що проходять через функцію втрат (cost function).

– Частинна похідна помилки обчислюється по кожній вазі (ці часткові диференціали відбивають внесок кожної ваги в загальну помилку (total_loss)).

– Потім ці диференціали множаться на число, називане швидкість навчання або learning rate (η).

Отриманий результат потім віднімається з відповідних ваг.

У результаті вийдуть наступні оновлені ваги:

$$w1 = w1 – (\eta * \partial(err) / \partial(w1))$$

$$w2 = w2 – (\eta * \partial(err) / \partial(w2))$$

$$w3 = w3 – (\eta * \partial(err) / \partial(w3))$$

Те, що ми припускаємо й ініціалізуємо ваги випадковим образом, і вони будуть давати точні відповіді, звучить не цілком обґрунтовано, проте, працює добре.

Якщо ви знайомі з рядами Тейлора, зворотне поширення помилки має такий же кінцевий результат. Тільки замість нескінченного ряду ми намагаємося оптимізувати тільки його перший член.

Зсуви – це ваги, додані до схованих шарів. Вони теж випадковим образом ініціалізуються й обновляються так само, як схований шар. Роль схованого шару полягає в тому, щоб визначити форму базової функції в даних, у те час як роль зсуву – зрушити знайдену функцію убік так, щоб вона частково збіглася з вихідною функцією.

Часткові похідні

Часткові похідні можна обчислити, тому відомо, який був внесок у помилку по кожній вазі. Необхідність похідних очевидна. Представте нейронну мережу, що намагається знайти оптимальну швидкість безпілотного автомобіля.

Якщо машина виявить, що вона їде швидше або повільніше необхідної швидкості, нейронна мережа буде міняти швидкість, прискорюючи або сповільнюючи автомобіль. Що при цьому прискорюється/уповільнюється? Похідні швидкості.

Гіперпараметри

Нейронна мережа використовується для автоматизації відбору ознак, але деякі параметри настроюються вручну.

Швидкість навчання (learning rate)

Швидкість навчання є дуже важливим гіперпараметром. Якщо швидкість навчання занадто мала, то навіть після навчання нейронної мережі протягом тривалого часу вона буде далека від оптимальних результатів. З іншого боку, якщо швидкість навчання занадто висока, то мережа дуже швидко видасть відповіді.

Функція активації (activation function)

Функція активації – це один із самих потужних інструментів, який впливає на силу, приписувану нейронним мережам. Почасти, вона визначає, які нейрони будуть активовані, іншими словами і яка інформація буде передаватися наступним шарам.

Без функцій активації глибокі мережі втрачають значну частину своєї здатності до навчання. Нелінійність цих функцій відповідає за підвищення ступеня волі, що дозволяє узагальнювати проблеми високої розмірності в більш низьких вимірах.

Функція втрати (loss function)

Функція втрат перебуває в центрі нейронної мережі. Вона використовується для розрахунків помилки між реальними й отриманими відповідями. Наша глобальна мета – мінімізувати цю помилку. Таким чином, функція втрат ефективно наближає навчання нейронної мережі до цієї мети.

Функція втрат вимірює «наскільки гарна» нейронна мережа у відношенні даної навчальної вибірки й очікуваних відповідей. Вона також може залежати від таких змінних, як ваги й зсуву.

Функція втрат одномірна й не є вектором, оскільки вона оцінює, наскільки добре нейронна мережа працює в цілому.

Деякі відомі функції втрат:

- Квадратична (середньоквадратичне відхилення).
- Крос-Ентропія.
- Експонентна (Adaboost).
- Відстань Кульбака-Лейблера або приріст інформації.

Середньоквадратичне відхилення – найпростіша функція втрат і найбільше часто використовувана.

Функція втрат у нейронній мережі повинна задовольняти двом умовам:

- Функція втрат повинна бути записана як середнє.
- Функція втрат не повинна залежати від яких-небудь активаційних значень нейронної мережі, крім значень, видаваних на виході.

Глибокі нейронні мережі

Глибоке навчання (deep learning) – це клас алгоритмів машинного навчання, які вчаться глибше (більш абстрактно) розуміти дані.

Більш формально в deep learning:

- Використовується каскад (пайплайн, як послідовно переданий потік) з множині обробних шарів (нелінійних) для добування й перетворення ознак.
- Ґрунтується на вивченні ознак (виставі інформації) у даних без навчання із вчителем. Функції більш високого рівня (які перебувають в останніх шарах) виходять із функцій нижнього рівня (які перебувають у шарах початкових шарах).
- Вивчає багаторівневі представлення, які відповідають різним рівням абстракції. рівні утворюють ієрархію представлення.

Не слід плутати із широкою нейронною мережею.

У цьому випадку велика кількість нейронів в одному шарі не приводить до глибокого розуміння даних. Але це приводить до вивчення більшого числа ознак.

Дуже заманливо використовувати глибокі й широкі нейронні мережі для кожного завдання. Але це може бути поганою ідеєю, тому що:

– Обидві вимагають значно більшої кількості даних для навчання, щоб досягтися мінімальної бажаної точності.

– Обидві мають експонентну складність.

– Занадто глибока нейронна мережа спробує зламати фундаментальні представлення, але при цьому вона буде робити помилкові припущення й намагатися знайти псевдо-залежності, які не існують.

– Занадто широка нейронна мережа буде намагатися знайти більше ознак, чим є. Таким чином, що подібно попередній, вона почне робити неправильні припущення про дані.

Прокляття розмірності

Прокляття розмірності ставиться до різних явищ, що виникають при аналізі й організації даних у багатомірних просторах (часто із сотнями або тисячами вимірів), і не зустрічається в ситуаціях з низькою розмірністю.

Граматика англійської мови має величезна кількість атрибутів, що впливають на неї. У машинному навчанні ми повинні представити їхніми ознаками у вигляді масиву/матриці кінцевої й суттєво меншої довжини (чому кількість існуючих ознак). Для цього мережі узагальнюють ці ознаки. Це породжує дві проблеми:

– Через неправильні припущення з'являється зсув. Високий зсув може привести до того, що алгоритм пропустить істотний взаємозв'язок між ознаками й цільовими змінними. Це явище називають недонавчання.

– Від невеликих відхилень у навчальній множині через недостатнє вивчення ознак збільшується дисперсія. Висока дисперсія веде до перенавчання, помилки сприймаються як надійна інформація.

Компроміс

На ранній стадії навчання зсув великий, тому що вихід з мережі далекий від бажаного. А дисперсія дуже мала, оскільки дані має поки малий вплив.

Наприкінці навчання зсув невеликий, тому що мережа виявила основну функцію в даних. Однак, якщо навчання занадто тривале, мережа також вивчить шум, характерний для цього набору даних. Це приводить до великого розкиду результатів при тестуванні на різних множинах, оскільки шум міняється від одного набору даних до іншого.

Дійсно, алгоритми з більшим зсувом звичайно в основі більш простих моделей, які не схильні до перенавчання, але можуть недонавчитися й не виявити важливі закономірності або властивості ознак. Моделі з маленьким зсувом і великою дисперсією звичайно більш складні з погляду їх структури, що дозволяє їм більш точно представляти навчальний набір. Однак вони можуть відображати багато шуму з навчального набору, що робить їхні прогнози менш точними, незважаючи на їхню додаткову складність.

Отже, як правило, неможливо мати маленький зсув і маленьку дисперсію одночасно.

Зараз є множин інструментів, за допомогою яких можна легко створити складні моделі машинного навчання, перенавчання займає центральне місце. Оскільки зсув з'являється, коли мережа не одержує досить інформації. Але чим більше прикладів, тим більше з'являється варіантів залежностей і зміностей у цих кореляціях.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для моделювання ботнет мережі, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

На сьогоднішній день вибір надійного антивірусу – справа першочергової важливості для будь-якого користувача, адже що може захистити комп'ютер краще, чим повноцінний комплексний застосунок, спрямований на схоронність даних і стабільну роботу? Цей розділ присвячений кращим антивірусним програмам, які дадуть користувачеві всі необхідні інструменти для захисту від троянів і вірусів, запобіжать зараженню комп'ютера, заблокують шкідливий сайт і будуть не сильно завантажувати систему.

Ми вже провели загальний аналіз самих популярних антивірусів 2021 року, з'ясувавши всі їх плюси й мінуси. Результати цієї перевірки ви можете подивитися в порівняльній таблиці. Ну а щоб визначити найкращий безкоштовний антивірус, давайте розглянемо їх більш детально. Отже, приступимося!

Avast Free Antivirus

Avast Free Antivirus – відмінний безкоштовний антивірус, який заслужив визнання мільйонів користувачів по усьому світу завдяки надійному захисту від троянів і вірусів у реальному часі. Остання версія Avast може похвастатися оновленим інтуїтивно зрозумілим інтерфейсом, декількома унікальними функціями (Autosandbox, Intelligent Scanner і т.д.), поліпшеною швидкодією й, головне, однієї із самих широких баз вірусів у світі (вона щодня поповнюється).

Ключові особливості Avast Free Antivirus:

- Величезна, постійно обновлювана база сигнатур.
- Відмінний захист від руткітів у реальному часі.
- Мережний екран, що захищає комп'ютер під час інтернет-серфінгу.

- Сучасний движок, що забезпечує завидну швидкодію програми.
- Автоматичний і ігровий режими роботи.
- Браузер Safezone для безпечного серфінгу.
- Зручний віджет на робочий стіл.
- Інтуїтивний, приємний ока інтерфейс.
- Безкоштовна версія дає повний функціонал.

Avast Free Antivirus представляє користувачеві нову функцію Autosandbox, яка дозволяє автоматизувати процес приміщення підозрілих файлів в "пісочницю", де можна буде провести повний аналіз файлу й, при необхідності, вилікувати його. Ця функція дозволяє врятувати від миттєвого стирання досить великий відсоток файлів, уникнути системних помилок, пов'язаних з видаленням важливих системних файлів і т.п. Застосунок поводиться з об'єктами акуратніше аналогів.

Також нова версія Avast містить у собі вбудовану функцію віддаленої підтримки. Користувач може підключитися до комп'ютера іншого користувача (тільки з дозволу) і виявити йому технічну підтримку або допомогу, що досить зручно, тому що рятує про необхідність мати на комп'ютері налаштовану програму для дистанційного доступу. У цілому, Avast Free Antivirus є відмінним вибором для середньостатистичного користувача, надаючи йому все необхідне для змісту системи в чистоті.

AVG Anti-Virus Free

AVG Anti-Virus Free – популярний антивірус основною характерною рисою якого є глибока інтеграція в систему. Він автоматично сканує файли й програми при їхньому запуску, що дозволяє уникнути зараження вірусами, троянами й шпигунськими програмами. Також AVG надає користувачеві сканер, що настроюється за розкладом. Завдяки цій функції ви самі зможете контролювати як процес перевірки комп'ютера на заражені файли, так і процес їх лікування. У новій версії AVG повністю оновлений інтерфейс, який тепер може похвастати приємним зовнішнім виглядом і зручними меню.

Ключові особливості AVG Antivirus Free:

- Швидке і якісне сканування системи.
- Автоматичне сканування файлу при його першому запуску.
- Сканер на вимогу / розкладу.
- Постійні відновлення.
- Корисні модулі захисту (Link Scanner, e-mail Scanner).
- Інтуїтивний інтерфейс.

AVG Anti-Virus Free може похвастатися відмінними показниками захисту системи й досить малим споживанням системних ресурсів. На відміну від платної версії, яка, звичайно, містить у собі більше інструментів і функцій, версія для безкоштовного використання працює набагато більш стабільно, одержуючи при цьому ту ж саму технічну підтримку у вигляді відновлень. Швидкість роботи програми вражає, а сканер електронної пошти позбавить вас від необхідності встановлювати спеціалізовані додатки, тому що справляється він на всі 100%.

Нова унікальна функція – Link Scanner дозволяє користувачеві використовувати антивірус щоб просканувати сайт не заходячи на нього, що може бути дуже зручно. Саме AVG Anti-Virus Free є вибором "по-умовчанню" для більш ніж 5 мільйонів користувачів по усьому світу, а нам залишається лише помітити, що це цілком заслужено.

Advanced Systemcare Ultimate з Антивірусом

Advanced Systemcare Ultimate – це засіб очищення й оптимізації системи, що включає потужний антивірусний сканер. На відміну від звичайних брендових антивірусників і вбудованого Захисника Windows 10, застосунок виявляє винятково позитивний вплив на продуктивність ПК. Воно поєднує дві самі потрібні функції – оптимізацію й антивірусний захист. За що й одержало настільки високе місце в рейтингу.

Крім того, Systemcare пропонує користувачеві велика кількість додаткових функцій: відновлення драйверів, менеджер паролів, резервне копіювання файлів, менеджер програм для групового видалення, сканування знімних носіїв,

очищення й виправлення реєстру, пошук великих, сміттєвих файлів, дублікатів. Говорячи простіше, цей застосунок для повноцінного адміністрування ПК із високим рівнем захисту особистих даних.

Особливості Advanced Systemcare Ultimate:

– Очищення, оптимізація й забезпечення інформаційної безпеки силами одного застосунку;

– Повноцінний захист від фішінгу, шкідливих спливаючих вікон в Chrome, Firefox і інших браузерах;

– Робота в реальному часі з можливістю відкладеного сканування зазначених об'єктів, доступний ігровий режим;

– Повне адміністрування ПК під керуванням Windows, позитивний вплив на його швидкодію;

– Дуже дешева річна підписка – навіть Bitdefender коштує набагато дорожче, при цьому гірше визначає комп'ютерні віруси.

Systemcare Ultimate придасться як власникам слабких машин, так і користувачам потужних комп'ютерів, тому що дозволить добитися максимальної продуктивності. А завдяки вбудованій системі виявлення вірусів на основі движка Bitdefender захистить від усіх актуальних погроз.

Panda Antivirus Pro

Panda Antivirus Pro служить однієї єдиної цілі – вона захищає комп'ютер користувача від найбільш відомих видів віртуальних погроз. І можна із упевненістю сказати, що справляється вона із цим чудово. Установивши Панду користувач одержує у своє розпорядження вкрай простий, але досить ефективний щит від будь-якої віртуальної погрози.

Досить більша база вірусів Панди постійно поповнюється як розроблювачами, так і користувачам, яким "везе" знаходити нові різновиди вірусів. Ну а в елементарному інтерфейсі цього безкоштовного антивірусу Panda Dome розбереться навіть дитина.

Ключові особливості Panda Antivirus Pro:

- Автоматично виявляє шкідливе програмне забезпечення.
- Блокує шкідливі сайти.
- Обновляє базу вірусів практично щодня.
- Спеціальні режими роботи для ігор і відтворення мультимедіа.
- Антируткіт файєрвол.
- Автоматично сканує, що підключаються USB пристрою.
- Інтуїтивно-зрозумілий інтерфейс Panda Cloud Antivirus.

Panda Antivirus Pro відмінний вибір для користувачів, які прагнуть одержати якісний захист від вірусів при мінімумі зусиль. Більшість функцій Панди автоматизовані, програма постійно сканує оперативну пам'ять і жорсткий диск на наявність погроз і підозрілих файлів. Новий движок застосунку дозволяє знизити споживання програмою системних ресурсів до мінімуму.

Антивірусний завантажувальний диск Panda Cloud Cleaner дає можливість вилікувати заражену систему, яка не може сама завантажитися. Може небагато напружувати кількість неправильних спрацьовувань, але ж це не так і погано – програма опікується про Вас! Загалом, якщо Вам потрібний відмінний антивірус, який гарно справляється з підтримкою системи в гарному стані навіть без участі користувача в цьому процесі, Panda Antivirus Pro – кращий вибір!

Iobit Malware Fighter

Iobit Malware Fighter не є класичним продуктом, але гарантують більший ступінь захисту, чому dr.web cureit і інші сканнери, розраховані на звичайну перевірку ПК на віруси. Також він може бути встановлений у комплекті із програмним забезпеченням Advanced Systemcare – набором утиліт, які очищають систему, підвищують продуктивність комп'ютера, відновлюють випадково віддалену інформацію і т.д. Найдеться засіб для застосунку ледве чи не будь-якої проблеми.

Ключові особливості Iobit Malware Fighter:

- Споживає мало апаратних ресурсів, має зрозумілий інтерфейс.

– Включає сигнатури виявлення троянських програм від Bitdefender.

– Містить відразу кілька модулів для захисту користувача під час роботи в Мережі.

– Анти-здирник допомагає зберегти конфіденційні дані й документи в неприступності для вредоносів.

– Дозволяє застосовувати дії до інфікованих об'єктів у ручному й автоматичному режимі.

– Пропонує також установити непоганий набір компонентів для очищення й налаштування ОС Windows.

– У порівнянні із платними аналогами, у базовій комплектації, має досить непоганий рівень, достатній для побутового використання.

Застосунок потрапив до нас в огляд, оскільки відмінно працює на слабких комп'ютерах, не вимагає плати за використання, однак, надає винний рівень захисту підключення до інтернету. Інші схожі продукти діють інакше: наголошують на сканування диска, забуваючи про можливість зараження через потенційно небезпечні сайти й різних шпигунських модулів. У якості домашнього антивірусника, Malware Fighter упорається із завданням, але для захисту масивів архі-важливих даних краще використовувати серйозні продукти від західних розроблювачів.

360 Total Security

360 Total Security – потужний набір інструментів для підтримки операційної системи в порядку, який містить у собі сучасний антивірус, твікер для оптимізації й інструмент для очищення системи від сміття. Це безкоштовний антивірусний застосунок, який здатний не тільки якісно захистити комп'ютер від зовнішніх погроз, але також і оптимізувати його роботу, допомогти правильно розподілити системні ресурси, щоб побільшати швидкість процесів. Сам застосунок базується на п'ятьох активних движках, чотири з яких відповідають за захист систему, тому можете бути впевнені, якість 360 Total Security повністю відповідає його назві.

Ключові особливості 360 Total Security:

- Відмінний захист від вірусів як у реальному часі, так і при скануванні.
- Використання декількох окремих модулів для захисту.
- Автоматична перевірка носіїв, що підключаються, інформації.
- Зручна інтеграція в браузери.
- Очищення системи від сміттєвих і тимчасових файлів.
- Чудова оптимізація системи.
- Абсолютно безкоштовна версія.

360 Total Security Антивірус є відмінним вибором як для починаючих користувачів ПК, так і для просунутих. Перші одержують у своє розпорядження надійну систему з множиною автоматизованих функцій, що дозволить захищати комп'ютер без прямої участі користувача.

Другі ж по гідності оцінять гнучкі налаштування застосунку, можливість міняти профілі, зберігаючи в них різні налаштування, функції по оптимізації роботи системи й багато інші цікаві опції. Оформлення застосунку не викликає ніяких питань і дозволяє використовувати всі її аспекти без зайвих питань і допомоги довідки. Захистите свій комп'ютер разом з 360 Total Security!

ESET NOD32 Smart Security

ESET NOD32 Smart Security – чудовий комплексний застосунок для захисту вашого комп'ютера від різного роду віртуальних погроз. Віруси, трояни, руткіти, рекламне ПЗ, спам – усе це легко забувається після установки цього чудового антивірусу. "Кращий захист – це напад", – імовірне вважають програмісти ESET, тому що за замовчуванням на NOD32 виставлені досить агресивні налаштування по скануванню й знищенню погрози. Але поки це дає такі результати – а чому б і ні?

Ключові особливості ESET NOD32 Smart Security:

- Тотальний багаторівневий захист від вірусів, malware і adware додатків.
- Персональний файервол.
- Захист від ботнетів і поліпшений блокувальник експлойтів.

– Функція Anti-Theft, яка дозволяє знайти й повернути загублений ноутбук.

– Smart Mode, що автоматизує процеси сканування й виявлення підозрілих файлів.

– Можливість створення завантажувального диска для ушкодженої системи.

– Мінімальний відсоток неправильних спрацьовувань.

– Симпатичний мінімалістичний інтерфейс.

– 30 днів повної робочої версії.

ESET NOD32 Smart Security має у своєму арсеналі все необхідне для захисту вашого ПК: кілька щаблів захисту від будь-якого типу небажаного ПЗ або вірусу, персональний кастомизуємий файєрвол для шифровки з'єднання, батьківський контроль, контроль і скан пристроїв, що підключаються, безкоштовна цілодобова техпідтримка й т.д. Якщо необхідний антивірус для установки на ноутбук, NOD32 буде практично ідеальним застосунком, тому що він має спеціальні профілі для роботи на портативних ПК, які дозволяють знизити витрата енергії.

Але за все доводиться платити, і в ESET NOD 32 крім величезного набору якісних інструментів також досить високе споживання системних ресурсів, що частково компенсується профілями, де можна настроїти кожний аспект. До слова, на офіційному сайті антивірусу є багато корисної інформації з оптимізації роботи застосунку. Але в цілому, ESET NOD32 Smart Security по праву є однієї із самих популярних і надійних антивірусних програм на сучасному ринку.

Microsoft Security Essentials

Microsoft Windows Defender – безкоштовний антивірусний софт від компанії Майкрософт, передвстановлений на комп'ютерах з Windows 10, однак може бути також інстальований на більш ранні версії операційної системи, де буде називатися Microsoft Security Essentials. Має досить просту функціональність і низькі системні вимоги. Його головні функції – видалення

шкідливих програм, сигнатурний аналіз файлів, завантажених з інтернету, можливість захисту Windows від злому як з боку користувача, так і з боку зловмисників.

Особливості Захисника Windows:

- Простий зручний інтерфейс антивірусного програмного забезпечення.
- Виявлення уразливостей хробаків шпигунів шкідливого коду скриптів зловредів фішінгових атак.
- Можливість окремо перевіряти файли вмісту різних накопичувачів – флешки, диска, зовнішнього HDD.
- Не встановлюється на Mac OS і старі складання Windows, підтримка яких завершена.

Avira Free Antivirus

Avira Antivirus – простий антивірус, який може похвастатися досить ефективним захистом від вірусів, троянів і рекламного ПЗ. Основною його перевагою над конкурентами є відхід у хмарну технологію, яка дозволяє захищати комп'ютер від самих нов, що з'явилися зовсім недавно, погроз. Сам антивірус надає лише базовий захист від погроз, але може бути розширений спеціальними модулями-плагінами, які можна скачати абсолютно безкоштовно із сайту виробника. У такий спосіб кожний користувач може "побудувати" персональну й унікальну систему захисту.

Ключові особливості Avira Free Security Suite:

- Постійно поповнювана антивірусна база.
- Використання хмарних технологій для економії системних ресурсів.
- Уміє боротися з макровірусами й лікувати заражені їм файли.
- Можливість налаштування сканування за розкладом.
- Автоматичне сканування файлів, що виконуються.
- Можливість довантажувати модулі, щоб розширити функціонал.
- Не конфліктує з іншими антивірусними застосунками.

– Випускається також комплексний застосунок Avira Prime, що представляє собою симбіоз захисного софта, VPN, генератора паролів.

Avira Antivirus Free іде в комплекті з модулем Virus Guard, який автоматично сканує файли, які відкриває користувач, що підвищує рівень безпеки системи. Для забезпечення безпеки в мережах wi-fi також можна активувати компонент захисту Avira Phantom VPN, необхідний для захисту ПК від фішінгових сайтів і вірусних погроз ззовні.

У цілому, Avira – одне із кращих застосунків для боротьби з так званими "поліморфними" вірусами, які можуть наслідувати звичайні програми. Також Ви можете встановити модуль сканування електронної пошти, захист від спама й від програм з автодозвоном. Іншими словами, Avira – гарний антивірус, який готовий довго й надійно захищати ваш комп'ютер, що б ви не робили.

Bitdefender Antivirus Free Edition

Bitdefender Antivirus Free Edition – безкоштовна версія популярного антивірусу Bitdefender, який ще може конкурувати з лідерами ринку антивірусних додатків, тому що з легкістю відловлює всі існуючі віруси.

У безкоштовну версію ввійшов сучасний сканер з календарним модулем, що дають можливість планувати перевірки системи наперед, карантинний модуль для спостереження за підозрілими файлами, журнал звітів про перевірку й величезна база вірусів, яка містить на сьогоднішній день близько 500 000 сигнатур. Існує думка, що безкоштовна версія ніколи не надасть того ж якості, яка є в платних антивірусних програмах, і Bitdefender з легкістю валить усі подібні стереотипи.

Ключові особливості Bitdefender Antivirus Free Edition:

- Надійний захист від вірусів і троянів.
- Простій у використанні.
- Зручне блокування шкідливих сайтів.
- Використовує рекордно мало системних ресурсів.
- Не гальмує систему.

– Запускається на Windows XP, Windows 7, Windows 8 і Vista.

Bitdefender Antivirus Free Edition ідеально підходить для домашнього використання, тому що не навантажує систему зайвими процесами, забезпечуючи при цьому високий рівень захисту. На відміну від багатьох інших антивірусів, Bitdefender не напружує, що постійно вискакують вікнами й не вимагає участі користувача у своїй роботі.

Величезна база вірусів забезпечує найвищий рівень захисту, а інтуїтивний інтерфейс дозволяє використовувати його навіть користувачам, які ніколи раніше не зустрічалися з антивірусними програмами. Спробуйте цей чудовий антивірус у дії й переконаєтеся, що це один із кращих представників на ринку!

Також існує версія Bitdefender Internet Security. Вона має вбудований брандмауера, який перевіряє трафік і захищає операційну систему від мережних атак, несанкціонованого доступу ззовні й інших кіберзагроз.

Comodo Antivirus

Comodo Antivirus – потужний безкоштовний антивірус для комплексного захисту комп'ютера від вірусів, троянів, хакерських атак і іншого шкідливого ПЗ. Завдяки розширеному евристичному аналізу файлів, Comodo дуже гарно справляється з виявленням заражених файлів, дозволяючи вилікувати їх швидше, чим вони завдадуть шкоди системі. Установити антивірус ще простіше, чим ним користуватися – по ходу установки Вам буде запропоновано вибрати множин налаштувань, щоб полегшити роботу із програмою після установки.

Ключові особливості Comodo Antivirus:

– Більша база вірусів.
– Вбудований календар для автоматизації сканування.
– Кращі показники евристичного аналізу серед конкурентів.
– Зручна ізоляція підозрілих файлів у карантин.
– Швидка і якісна техпідтримка.
– Практично ідеальна проактивний захист.
– Симпатичний дизайн застосунку Comodo Free Internet Security.

Comodo Internet Security відмінно підійде як новачкам, так і просунутим користувачам, тому що вдало поєднує в собі багатий інструментарій, зручні налаштування по автоматизації процесів і має приємний інтерфейс.

Новітній движок Comodo дозволяє практично не завантажувати системи під час роботи, а також скорочує стандартно довгий час очікування під час, приміром, процесу сканування комп'ютера на погрози. У підсумку ми маємо прекрасний кастомізований антивірус, який буде вірно служити багато років.

Dr.Web Antivirus

Dr.Web Antivirus – безкоштовний антивірусний застосунок для виявлення й знищення вірусів і іншого шкідливого ПЗ. Завдяки ефективному евристичному аналізатору, Dr.Web легко виявляє нові невідомі види віртуальних погроз навіть у соціальних мережах, а проактивний багатоступінчастий захист обгороджує систему від будь-якої небезпеки під час інтернет серфінгу або використання неперевірених носіїв інформації. Складанням база займаються незалежні лабораторії що підвищує якість бази вірусів.

Ключові особливості Dr.Web Antivirus:

- Багатоступінчаста система захисту.
- Модулі для сканування USB-носіїв, електронної пошти й т.д.
- Захист користувацьких даних від ушкодження.
- Висока швидкість противірусного сканування.
- Персональний мережний екран для захисту від хакерів.
- Не сповільнює роботу комп'ютера.
- Максимально спрощений інтерфейс із симпатичним дизайном.
- Безкоштовна пробна версія. Якщо ви розраховуєте на повністю безкоштовний застосунок, скачайте невимогливий маленький антивірус Dr.Web Cureit.

Dr.Web Antivirus здатний задовольнити запити навіть самого причепливого користувача: висока швидкодія застосунку дозволяє йому сканувати комп'ютер за лічені хвилини, активний мережний фільтр змушує

забути про будь-яку небезпеку на просторах Мережі, а зручний і інтуїтивно-зрозумілий інтерфейс буяє зручними меню, кнопками швидкого доступу й легко кастомізується. У цілому, антивірус Dr.Web має всі шанси стати Вашим незамінним помічником у справі змісту ПК у порядку.

AdwCleaner

AdwCleaner – невелика й безкоштовна утиліта, на відміну від повноцінних платних антивірусів вона створена для «одноразової» перевірки комп'ютера й відмінно підійде для аматорів завантаження контенту через сумнівні сайти. Інтелектуальне сканування дозволяє знайти не тільки шкідливі файли, але й програми, які встановилися без вашого ведена. Звичайно таке відбувається, коли в інсталятор додатково «вшивають» партнерське програмне забезпечення.

Особливості AdwCleaner:

- Видаляє ПЗ, установлене в пакету без вашої згоди.
- Очищення від троянів, шифрувальників, здирників і інших невідомих вірусів.
- Не може бути рекомендований у якості основного захисного ПЗ, краще віддати перевагу одному зі світових або російських антивірусів.

Багато описаних продуктів у безкоштовній версії відрізняються урізаним функціоналом. Найбільший відсоток працюючих нормально функцій надає Avast. Тут є й захист у режимі реального часу, і посилань, і антизлодій, і інструментарій зберігання паролів. Загалом, 99% необхідних функцій. Обмеження поширюється тільки на відключення спливаючих повідомлень. Програма періодично буде нагадувати купити повну версію, однак, тривалість показу банера біля годин можна зменшити до 1 секунди!

Вибираючи кращий безкоштовний антивірус 2021 року, ми не згадали й не представленняли оцінки для Norton Internet Security, Dr. Web Cureit, Trend Micro, Nano Antivirus, Tencent PC Manager, Adaware Antivirus Free, Zillya, Norton Security Deluxe, G Data, Sophos Home, Mcafee Internet Security, Emsisoft Anti-Malware, Bullguard Internet Security, Malwarebytes Anti-Malware, Endpoint Security. Також у

список антивірусів не ввійшли Zonealarm Free і відносно новий продукт китайських розроблювачів Qihoo 360. І нехай вбудований Захисник Windows провалює тестування ефективності антивірусів і не захищає від новітніх вірусів, програм-здирників, негативно впливає на продуктивність системи, його високе місце серед популярних антивірусних програм незаперечно.

Непогані показники також демонструє міжнародний досвід, що перейняв, Bitdefender (разом з його сигнатурами) Iobit Malware Fighter (компонент Advanced System Care). Реалізована як захист у фоновому режимі, так і непоганий файловий антивірус. Що до інших представлених у списку антивірусів – кожний з них здатний піймати шкідливу програму, виявити рекламне ПЗ, і вилікувати заражений файл. По великому рахункові, різниця в додаткових функціях типу захисту веб-камери й зручності використання, роботи зі статистикою.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000

виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена множин поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали множин декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для моделювання ботнет мережі.

В процесі розробки кваліфікаційної бакалаврської роботи необхідно виконати наступний обсяг роботи:

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

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

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

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

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

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

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

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

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Що ми маємо: сервер під DDoS середньеким ботнетом (20,000-50,000) і access.log'и з нього. Мати access.log до початку DDoS'a досить корисно, тому що він описує практично 100% легітимних клієнтів, а отже, відмінний dataset для тренування нейронної мережі.

Що ми прагнемо одержати: класифікатор, який по запису в access.log говорив би нам з деякої часток імовірності від кого прийшов запит: від бота або від людини. Завдяки такому класифікатору ми зможемо сформувати вектор атаки (набір підмереж) і відправити їх в firewall / хостеру.

Підготовка

Для тренування класифікатора нам знадобляться два набори даних: для "поганих" і "гарних" запитів. Так, досить забавно, що для того, щоб знайти ботів, потрібно спочатку знайти ботів. Отут нам саме й знадобляться grep, наприклад, для того, щоб витягтися з логів всі запити з IP'шников з більшим кіл-вом 503'їх помилок (rate limiting nginx'a).

У підсумку в нас повинні вийти 3 access.log'a:

- Dataset з "гарними" запитами з access.log'a до початку DDoS'a.
- Dataset з "поганими" запитами, нагрепаними на попередньому етапі.
- Dataset, який нам необхідно класифікувати, розділивши запити на погані й гарні. Звичайно це tail -f access.log'a із сервра під DDoS'ом.

Далі необхідно навчитися парсить т.зв. combined логи nginx'a.

Після того як ми навчилися парсить логи, необхідно виділити features (особливості/маркери/ознаки), скласти їхній словник і по ньому надалі становити feature-vector для кожного запису dataset'ов. Про це далі.

Машинне навчання в загальному

Ми зупинилися на виділенні features з балок, для цього беремо "гарний" і "поганий" логи й намагаємося знайти в них усілякі features aka ознаки. В combined балці їх не так вуж і багато, я побрав наступні:

– Сам запит, попарсенний на тип запиту(HEAD/GET/POST/etc), url і http_version. Url парситься на протокол, ім'я хоста, шлях і всі ключі від query_string

– Referer, попарсенний аналогічно url у запиті.

– User-Agent, попарсенний чарівною вуличною магією, тому що його формат досить сильно варіюється від браузера до браузера. В RFC2616 про нього сказане зовсім небагато. Напевно можна багато краще.

– status, тільки у випадку кодів 503/404/403. Взагалі, під час DDoS'a сервер любить відповідати 500/502, тому враховувати будемо тільки вищеописані коди.

Отже, ми підійшли до моменту, коли маємо на руках величезний список усіляких features, які можуть бути присутнім у запиті. Це і є наш словник. Словник потрібний для того, щоб з будь-якого можливого запиту створити feature-vector. Бінарний (що полягає з нулів і одиниць) M -Мірний вектор (де M – довжина словника), який відбиває присутність кожної ознаки зі словника в запиті.

Дуже добре, якщо словник з погляду структури даних буде hash-таблицею, тому що до нього буде множин обігів типу if word in dictionary.

Поділ Dataset'a

Гарною практикою є поділ dataset'a на кілька частин. Я бив на дві частини в пропорції 70/30:

– Training set. На ньому ми навчаємо нашу нейронну мережу.

– Test set. Їм ми перевіряємо, наскільки добре навчена наша нейронна мережа.

Така розбивка обумовлена тем фактом, що нейронна мережа з найменшим training error (помилкою на training set) буде видавати більшу помилку на нових даних, тому що ми «перенавчили» мережу, заточивши її під training set.

Надалі, якщо прийде спантеличитися вибором оптимальних констант, dataset треба буде розбити на 3 частині в співвідношенні 60/20/20: Training set, Test set і Cross validation. Останній саме й буде служити для вибору оптимальних параметрів нейронної мережі (наприклад weightdecay).

Нейронна мережа зокрема

Тепер, коли в нас на руках більше немає ніяких текстових балок, а є тільки матриці з feature-vector'ов, можна приступати до побудови самої нейронної мережі.

Почнемо з вибору структури. Я вибрав мережу з одного схованого шару розміром з подвоєний вхідний шар. Чому? Усі просто: так заповів Andrew Ng у випадку, якщо не знаєте із чого почати. Думаю, надалі із цим можна погратися, порисувавши графіки навчання.

Функцією активації для схованого шару обрана багатостраждальна сигмойда, а для вихідного шару – Softmax. Останній обраний на випадок, якщо прийде робити багатокласову класифікацію с mutually exclusive класами. Наприклад, "гарні" запити відправляти на бекенд, "погані" – у бан на файєрволі, а "сірі" – розгадувати капчу.

Нейронна мережа схильна до відходу в локальний мінімум, тому в себе в коді я ладу кілька мереж і вибираю ту, у якої найменший Test error (Помітьте, саме помилка на test set, а не trainig set).

Продуктивність

– lfu_cache. Портував з Activestate, щоб сильно прискорити обробку запитів-"високочастотників". Down-side – підвищене споживання пам'яті.

– Pybrai, не дуже швидкий, однак, він може використовувати Atlas-Based-Модуль arac, якщо при створенні мережі вказати Fast=True.

– Розпаралювання. Свою нейронну мережу я навчив на досить "товстому" серверному Nehalem'e, однак, навіть там відчувалася недостатність однопотокового навчання. Можна помізкувати на тему розпаралювання навчання нейронної мережі. Простий застосунок – тренувати відразу кілька нейронних мереж паралельно й з них вибирати кращу, але це створить додаткове навантаження на пам'ять, що теж не дуже добре. Хотівся б більш універсальний застосунок.

– Споживання пам'яті й кількість features. Гарною оптимізацією по пам'яті був перехід зі стандартних масивів на numpy'ние. Так само зменшення розміру dictionary і/або використання PCA може дуже добре допомогти, про це дещо нижче.

– Додаткові поля в логи. В combined логи можна додати ще багато всього, варто подумати на тему, які поля допоможуть в ідентифікації ботів. Можливо, має сенс ураховувати перший октет IP адреси, тому що в не інтернаціональному web-проекті китайські користувачі найімовірніше боти.

– Principal Component Analysis. Дозволить сильно скоротити розмірність feature-vector'ов, тим самим зменшивши час тренування нейронної мережі.

– Preprocessing/Normalizing для features. Наприклад, усе, що попадає під regexp [a-fa-f0-9]{32}, можна заміняти словом __MD5__, усе, що схоже на дату – словом __DATE__ і т.д., тим самим зменшити кіл-у малочастотних features і відповідно розмір словника.

– Тюнинг констант і структури нейронної мережі. Мало б зміст порисувати графіки навчання нейронної мережі при різних значеннях параметрів, благо matplotlib під рукою.

– Online навчання. Стратегія розумних атакуючих часто міняється. Добре б було придумати способи для до-/перенавчання нейронної мережі " на ходу". Втім, це легше сказати, чим зробити.

3.2 Розробка структурної схеми

Важливим фактором, що сприяють популярності використання ботнетів серед кіберзлочинців у наш час, є та відносна легкість, з якої можна зібрати, поміняти й удосконалити різні компоненти шкідливого програмного забезпечення ботнету. Можливість для швидкого створення ботнету з'явилася ще в 2015 році, коли в загальному доступі виявилися вихідні коди Lizardstresser, інструментарію для проведення DDoS-атак, створеного відомої хакерською групою Lizard Squad.

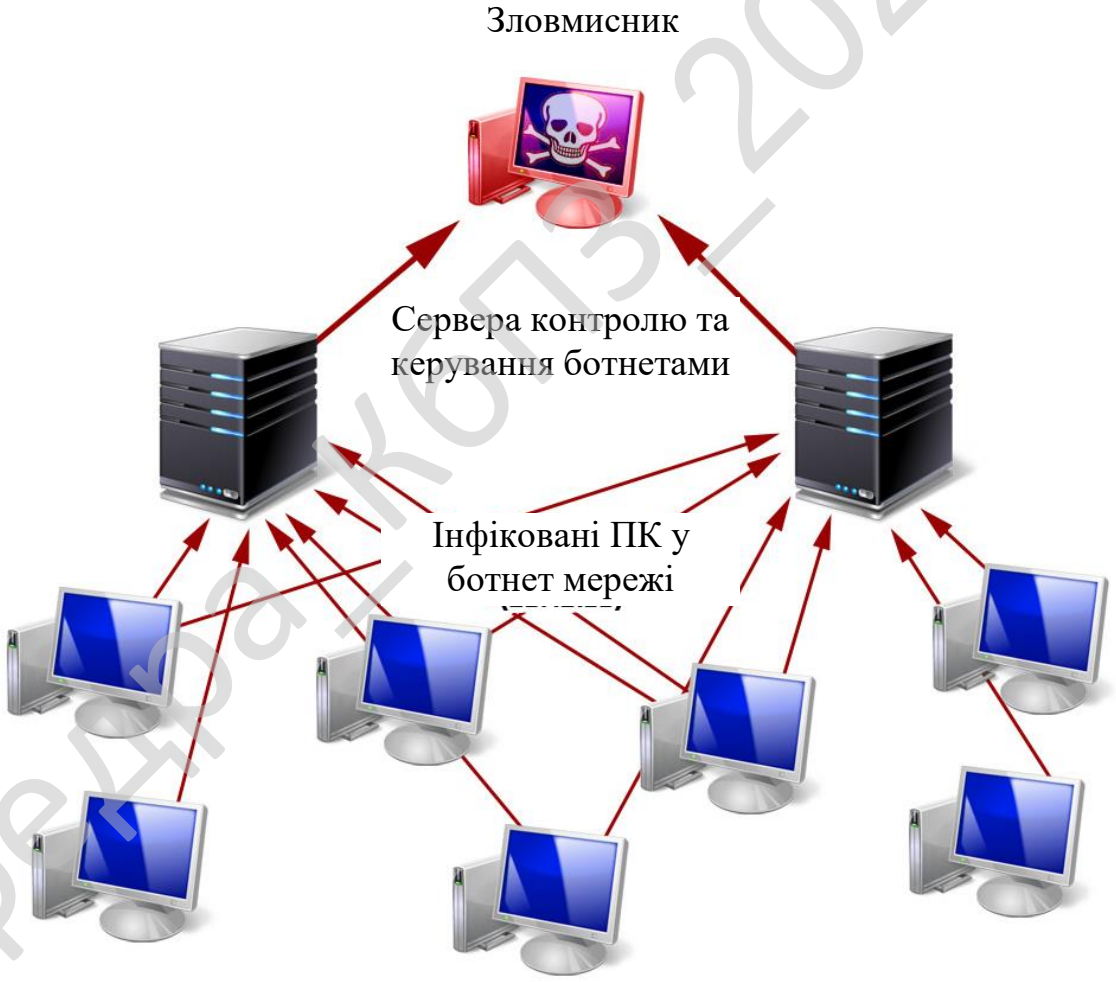

Рисунок 3.1 – Структурна схема системи

Легко доступний для завантаження й простіи у використанні код Lizardstresser містить деякі складні методи для здійснення DDoS-атак: тримати відкритим TCP-з'єднання, посилати випадкові рядки зі сміттєвим змістом символів на TCP-порт або UDP-порт, або повторно відправляти TCP-пакети із заданими значеннями прапорів. Шкідлива програма також включала механізм для довільного запуску команд оболонки, що є надзвичайно корисним для завантаження оновлених версій Lizardstresser з новими командами й оновленим списком контрольованих пристроїв, а також для установки на заражений пристрій іншого шкідливого програмного забезпечення. З тих пор були опубліковані вихідні коди й інших шкідливим програм для організації й контролю ботнетів, включаючи, у першу чергу, ПЗ Mirai, що драматично зменшило «високотехнологічний бар'єр» для початку кримінальної активності й, у той же час, побільшало можливості для одержання прибутку й гнучкості застосування ботнетів.

Інтернет речей (IoT) для створення ботнетів

З погляду кількості заражених пристроїв і генеруємого ними під час атак трафіка, вибухоподібний ефект мало масове використання незахищених IoT-пристроїв, що привело до появи безпрецедентним по своїх масштабах ботнетів. Так, прикладу, улітку 2016 року до й безпосередньо під час Олімпійських Ігор у Ріо-де-Жанейро один з ботнетів, створений на основі програмного коду Lizardstresser, в основному використовував порядку 10 тис. заражених IoT-пристроїв (у першу чергу – веб-камери) для здійснення численних і тривалих у часі DDoS-атак зі стійкою потужністю більш 400 Гбіт/с, що досяглася значення 540 Гбіт/с під час свого піка. Відзначимо також, що, згідно з оцінками, оригінальний ботнет Mirai зміг скомпрометувати близько 500 тис. IoT-пристроїв по усьому світу.

Незважаючи на те, що після подібних атак багато виробників внесли деякі зміни, IoT-пристрою здебільшого усе ще поставляються із передвстановленними заводськими налаштуваннями імені користувача й пароля або з відомими

уразливостями в безпеці. Крім того, щоб заощадити час і гроші, частина виробників періодично дублюють використовуване апаратне й програмне забезпечення для різних класів пристроїв. Як результат: паролі за замовчуванням, використовувані для керування вихідним пристроєм, можуть бути застосовані для множині зовсім інших пристроїв. Таким чином, мільярди незахищених IoT-пристроїв уже розгорнуті. І, незважаючи на те, що прогнозований ріст їх кількості сповільнився (хоч і незначно), очікуване збільшення світового парку «потенційно небезпечних» IoT-пристроїв у недалекому майбутньому не може не шокувати.

Багато IoT-пристроїв прекрасно підходять для неправомочного використання в складі злочинних ботнетів, тому що:

– Здебільшого вони некеровані, інакше кажучи, працюють без винного контролю з боку системного адміністратора, що робить їхнє застосування як анонімних проксі надзвичайно ефективним.

– Звичайно вони перебувають онлайн 24x7, а значить – вони доступні для здійснення атак у будь-який час, причому, як правило, без яких-небудь обмеження по пропускній здатності або фільтрації трафіка.

– Вони часто використовують урізану версію операційної системи, реалізовану на базі сімейства Linux. А шкідливе програмне забезпечення ботнетів може бути легко скомпільоване для широко використовуваних архітектур, в основному – ARM/MIPS/x86.

– Урізана операційна система автоматично означає менше можливостей для реалізації функцій безпеки, включаючи формування звітності, тому більшість погроз залишаються непоміченими власниками цих пристроїв.

От ще один недавній приклад, який допоможе усвідомити ту міць, якої можуть мати сучасні кримінальні інфраструктури ботнету: у листопаді 2017 року ботнет Necurs здійснив розсилання нового штаму вірусу-шифрувальника Scarab. У результаті масової компанії було відправлено близько 12,5 млн. інфікованих електронних листів, тобто швидкість розсилання склала більш ніж 2 млн. листів у

годину. До слова, цей же ботнет був помічений у поширенні банківських троянів Dridex і Trickbot, а також вірусів-здирників Locky і Jans.

Та, яка склалася в останні роки благодатна ситуація для кіберзлочинців, пов'язана з високою доступністю й простотою використання більш складного й гнучкого шкідливого програмного забезпечення для ботнетів у комбінації зі значним приростом кількості незахищених IoT-пристроїв, зробила кримінальні ботнети основним компонентом зростаючої цифрової підпільної економіки. У цій економіці є ринки для збуту отриманих нелегальним шляхом даних, здійснення шкідливих дій проти конкретних цілей у рамках надання послуг з наймання, і навіть для власної валюти. І всі прогнози аналітиків і фахівців з безпеки звучать украй невтішно – у недалекому майбутньому ситуація з неправомірним використанням ботнетів для одержання незаконного прибутку тільки погіршиться.

3.3 Розробка функціональної схеми

Ботнети являють собою комп'ютерні мережі, що полягають із великої кількості підключених до Інтернету комп'ютерів або інших пристроїв, на яких без ведена їхніх власників завантажене й запущене автономне програмне забезпечення – боти. Цікаво, що спочатку самі боти були розроблені як програмні інструменти для автоматизації некримінальних одноманітних і повторюваних завдань. По іронії долі, один з перших успішних ботів, відомий як Eggdrop, і створений в 1993 році, був розроблений для керування й захисту каналів IRC (Internet Relay Chat) від спроб сторонніх осіб захопити керування ними. Але кримінальні елементи швидко навчилися використовувати міць ботнетів, застосовуючи їх як глобальні, практично автоматичні системи, що приносять прибуток.

За цей час шкідливе програмне забезпечення ботнетів значно розвилося, і зараз може використовувати різні методи атак, які відбуваються одночасно по

декільком напрямкам. Крім того, «ботекономіка», з погляду кіберзлочинців, виглядає надзвичайно привабливо. Насамперед, практично відсутні витрати на інфраструктуру, тому що для організації мережі із заражених машин використовуються скомпрометовані комп'ютери й інше встаткування з підтримкою виходу в Інтернет, природно, без ведена власників цих пристроїв. Ця воля від вкладень в інфраструктуру означає, що прибуток злочинців буде фактично рівний їхньому доходу від незаконної діяльності. Крім можливості використовувати настільки «вигідну» інфраструктуру, для кіберзлочинців також надзвичайно важлива анонімність. Для цього при вимозі викупу вони, в основному, використовують такі «, що не відслідковуються» криптовалюти, як Bitcoin. Із цих причин ботнети стали найбільш предпочитаемою платформою для кіберкриміналу.

З погляду реалізації різних бізнес-моделей, ботнети є прекрасною платформою для запуску різної шкідливої функціональності, що приносить кіберзлочиникам незаконний дохід:

– Швидке й масштабне поширення електронних листів, що містять програми-здирники, що вимагають викуп.

– Як платформа для накручування числа кліків по посиланню.

– Відкриття проксі-серверів для анонімного доступу в Інтернет.

– Здійснення спроб злому інших інтернет-систем методом повного перебору (або « грубої сили»).

– Проведення масових розсилань електронних листів і здійснення хостингу підроблених сайтів при великомасштабному фішінгу.

– Відведення CD-ключів або інших ліцензійних даних на програмне забезпечення.

– Крадіжка персональної ідентифікаційної інформації.

– Одержання даних про кредитні картки й іншої інформації про банківський рахунок, включаючи PIN-коди або «секретні» паролі.

– Установка клавіатурних шпигунів для захвата всіх даних, які користувач уводить у систему.

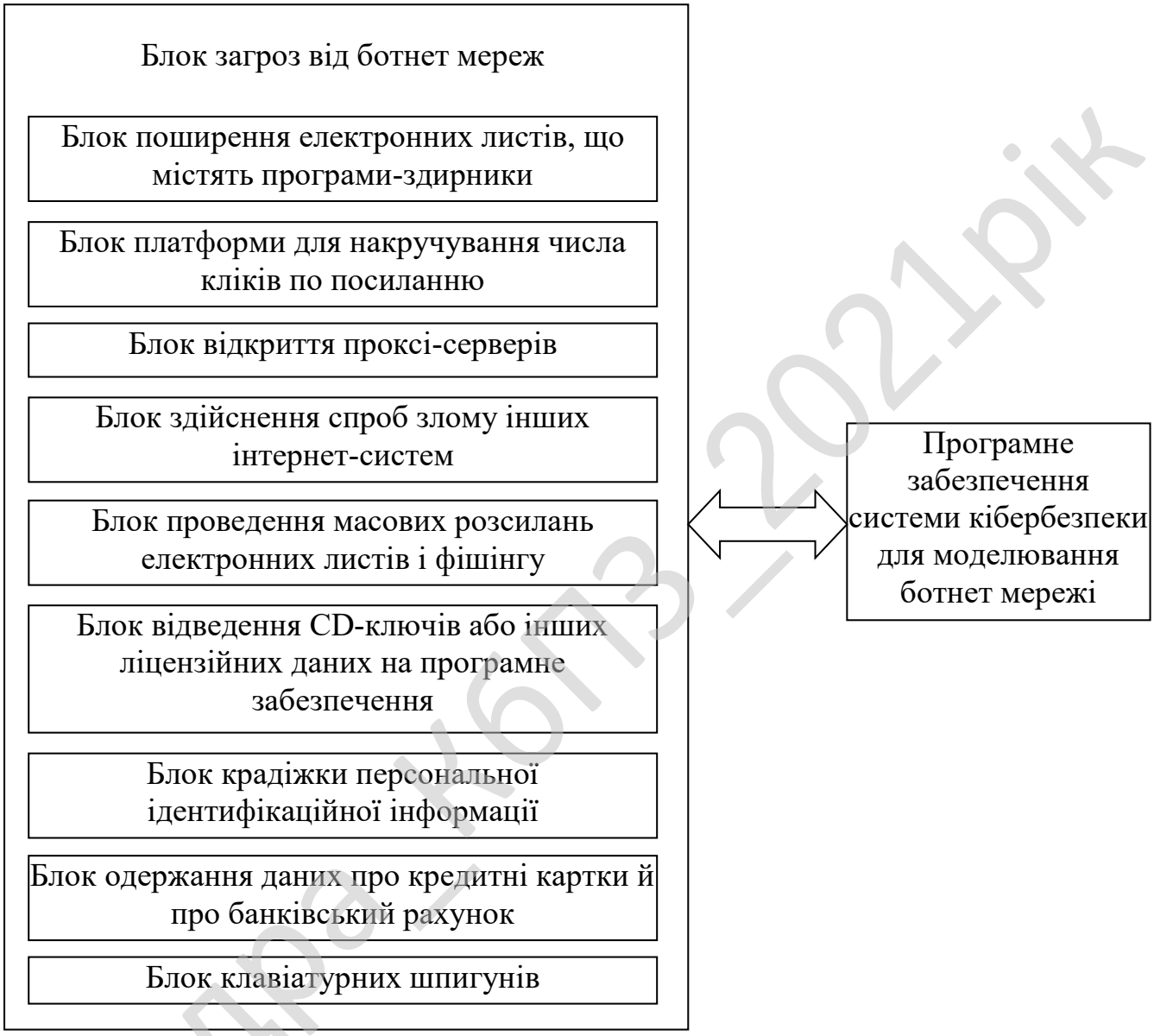

Рисунок 3.2 – Функціональна схема системи

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

Рисунок 3.3 – Діаграма взаємодії процесів

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, потік сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

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

– Потоки даних гібридні між елементами трьох попередніх типів.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Розглянемо блок-схеми системи та опис алгоритмів функціонування системи, спочатку розглянемо алгоритм роботи основної програми представлений у вигляді блок-схеми що зображено на рисунку 4.1.

З рисунку видно, що після запуску програми спочатку відбувається виведення вікна аналізатора мережі з відкриттям створеної нейронної мережі з послідуючим викликом підпрограми що зображено на рисунку 4.2.

У підпрограмі йде робота з нейромережею, а саме проходе ініціалізація, налаштування, введення даних, та робота з основним вектором.

Після чого у основній блок схемі проходе аналіз стану системи, виведення аналізу результату, з запитом та реалізацією функції роботи з бібліотекою шаблонів у окремому вікні. Далі йде запит завершення роботи системи та завершення циклу роботи.

При створенні системи кібербезпеки для моделювання ботнет мережі, в першу чергу було розроблено модуль роботи з моніторингом TCP/IP з'єднань робочої станції, це дозволило реалізувати весь функціонал системи у подальшому.

Розглянемо модуль uTCPIP що дозволяє проводити моніторинг TCP/IP з'єднань у вигляді вихідного коду:

```
unit uTCPIP;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, IPHelper, StdCtrls, ComCtrls, Buttons, IpHlpApi, jpeg;
```

```
        type
        // об’ява класу TIPForm з описом відкритих та закритих
// секцій та компонентів форми
          TIPForm = class( TForm )
            Timer1: TTimer;
            PageControl1: TPageControl;
            ARPSheet: TTabSheet;
            ConnSheet: TTabSheet;
            IP1Sheet: TTabSheet;
            ARPMemo: TMemo; // Поле введення текстових даних
            StaticText1: TStaticText; // Текст форми
            TCPMemo: TMemo; // Поле введення текстових даних
            StaticText2: TStaticText; // Текст форми
            IPAddrMemo: TMemo; // Поле введення текстових даних
            StaticText6: TStaticText; // Текст форми
            IP2Sheet: TTabSheet;
            IPForwMemo: TMemo; // Поле введення текстових даних
            StaticText8: TStaticText; // Текст форми
            AdaptSheet: TTabSheet;
            AdaptMemo: TMemo; // Поле введення текстових даних
            SpeedButton1: TSpeedButton;
            cbTimer: TCheckBox;
            StaticText9: TStaticText; // Текст форми
            NwMemo: TMemo; // Поле введення текстових даних
            StaticText10: TStaticText; // Текст форми
            TabSheet1: TTabSheet;
            ICMPInMemo: TMemo; // Поле введення текстових даних
            ICMPOutMemo: TMemo; // Поле введення текстових даних
            StaticText12: TStaticText; // Текст форми
            btRTTI: TSpeedButton;
            cbRecentIPs: TComboBox;
            edtRTTI: TEdit;
            StaticText14: TStaticText; // Текст форми
            IfMemo: TMemo; // Поле введення текстових даних
            StaticText15: TStaticText; // Текст форми
            TCPStatMemo: TMemo; // Поле введення текстових даних
            StaticText7: TStaticText; // Текст форми
            UDPStatsMemo: TMemo; // Поле введення текстових даних
            StaticText4: TStaticText; // Текст форми
            IPStatsMemo: TMemo; // Поле введення текстових даних
            StaticText5: TStaticText; // Текст форми
            UDPMemo: TMemo; // Поле введення текстових даних
```

Рисунок 4.1 – Блок-схема основної програми

```
StaticText3: TStaticText; // Текст форми
Panel1: TPanel; // Панель форми
Label1: TLabel; // Текст форми
Label2: TLabel; // Текст форми
Label3: TLabel; // Текст форми
procedure Timer1Timer( Sender: TObject );
procedure FormCreate( Sender: TObject );
procedure SpeedButton1Click( Sender: TObject );
procedure cbTimerClick( Sender: TObject );
```

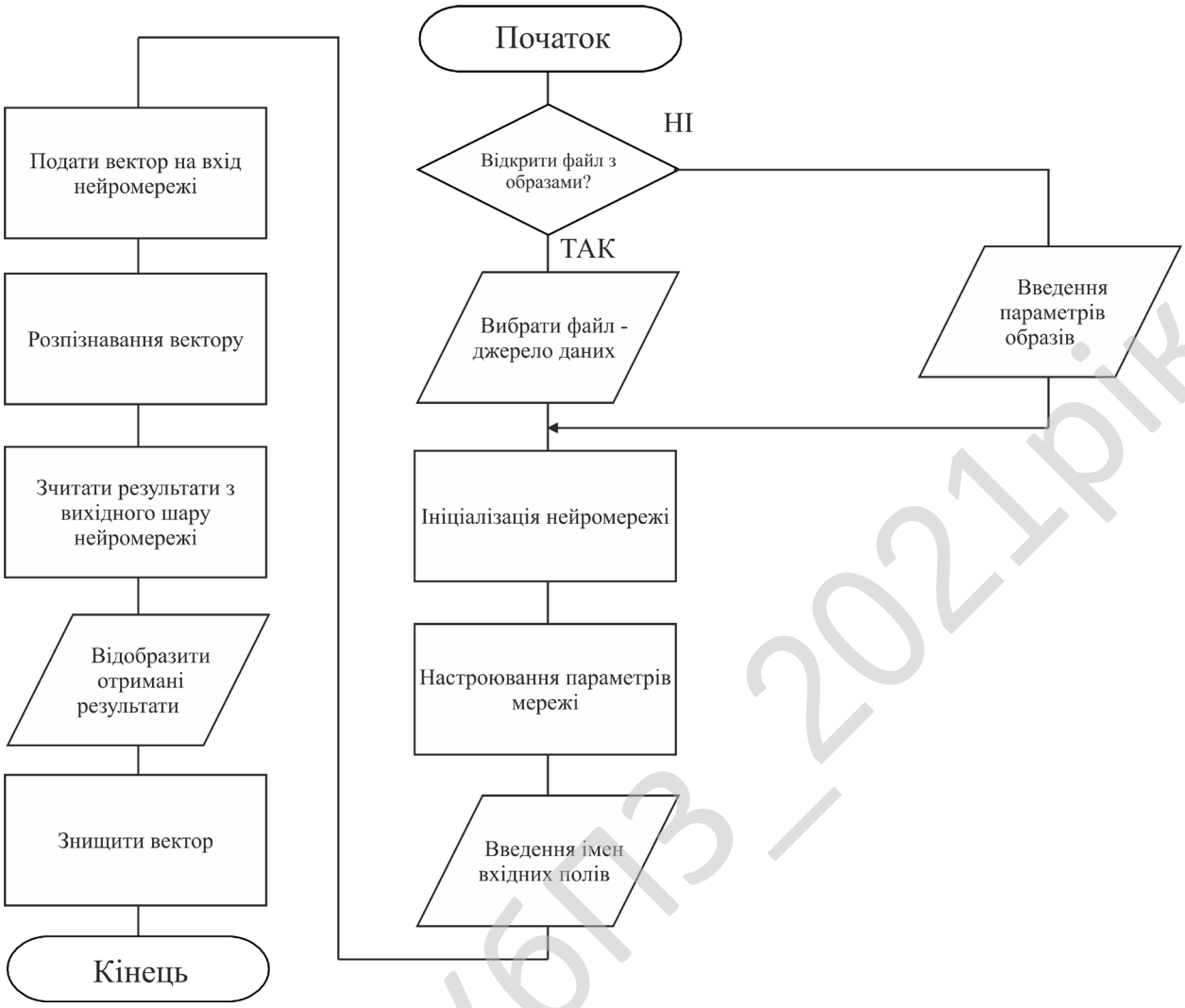

Рисунок 4.2 – Блок-схема роботи підпрограми

```
    procedure btRTTIClick( Sender: TObject );
    procedure cbRecentIPsClick( Sender: TObject );
  private
    { Private declarations }
    procedure DOIpStuff;
  public
    { Public declarations }
  end;

var // створення екземпляру класу TIPForm
  IPForm        : TIPForm;

implementation
```

```
{$R *.DFM}

//---------------------------
// Створення форми, дії конструктору
procedure TIPForm.FormCreate( Sender: TObject );
begin
  PageControl1.ActivePage := ConnSheet;
  Caption :=  Version;
  if LoadIpHlp then
  begin
      DOIpStuff;
      Timer1.Enabled := true;
  end
  else
      ShowMessage(' DLL бібліотека недоступна або не підтримується') ;
end;

//---------------------------
// Виклики мережних функцій, заповнення полів форми
procedure TIPForm.DOIpStuff;
begin
  Get_NetworkParams( NwMemo.Lines );
  Get_ARPTable( ARPMemo.Lines );
  Get_TCPTable( TCPMemo.Lines );
  Get_TCPStatistics( TCPStatMemo.Lines );
  Get_UDPTable( UDPMemo.Lines );
  Get_IPStatistics( IPStatsMemo.Lines );
  Get_IPAddrTable( IPAddrMemo.Lines );
  Get_IPForwardTable( IPForwMemo.Lines );
  Get_UDPStatistics( UDPStatsMemo.Lines );
  Get_AdaptersInfo( AdaptMemo.Lines );
  Get_ICMPStats( ICMPInMemo.Lines, ICMPOutMemo.Lines );
  Get_IfTable( IfMemo.Lines );
  Get_RecentDestIPs( cbRecentIPs.Items );
end;

//-------------------------------
procedure TIPForm.cbTimerClick( Sender: TObject );
begin
  Timer1.Enabled := (cbTimer.State = cbCHECKED) ;
end;
```

```
//-------------
// Дії циклічні таймеру 1
procedure TIPForm.Timer1Timer( Sender: TObject );
begin
  if cbTimer.State = cbCHECKED then
  begin
    Timer1.Enabled := false;
    DoIPStuff;
    Timer1.Enabled := true;
  end;
end;

//-------------
// Дії що реалізуються при натисненні на кнопку SpeedButton1Click
procedure TIPForm.SpeedButton1Click( Sender: TObject );
begin
  Speedbutton1.Enabled := false;
  DoIPStuff;
  Speedbutton1.Enabled := true;
end;

//-------------
procedure TIPForm.btRTTIClick( Sender: TObject );
var
  IPadr         : dword;
  Rtt, HopCount : longint;
  Res           : integer;
begin
  btRTTI.Enabled := false;
  Screen.Cursor := crHOURGLASS;
  IPadr := Str2IPAddr( edtRTTI.Text );
  Res := Get_RTTAndHopCount( IPadr, 128, RTT, HopCount );
  if Res = NO_ERROR then
    ShowMessage( ' Час в обидва кінці '
      + inttostr( rtt ) + ' ms, '
      + inttostr( HopCount )
      + ' hops to : ' + edtRTTI.Text
      )
  else
    ShowMessage('Виникла помилка:' + #13
                + ICMPErr2Str( Res ) ) ;
  btRTTI.Enabled := true;
```

```
  Screen.Cursor := crDEFAULT;
end;

//-------------
// обробка кліку форми
procedure TIPForm.cbRecentIPsClick( Sender: TObject );
begin
  edtRTTI.Text := cbRecentIPs.Items[cbRecentIPs.ItemIndex];
end;

end.
```

NetBIOS (Network Basic Input/Output System) – протокол для роботи в локальних мережах на персональних ЕОМ типу IBM/PC, розроблений у вигляді інтерфейсу, який не залежить від фірми–виробника. Був розроблений фірмою Sytek Corporation за замовленням IBM в 1983 році. Він включає в себе інтерфейс сеансового рівня (англ. NetBIOS interface), в якості транспортних протоколів використовує TCP і UDP.

Особливістю NetBIOS є можливість його роботи поверх різних протоколів, найпоширенішими/відомими з яких є NetBEUI, IPX і стек протоколів TCP/IP; причому якщо старі версії Windows орієнтувалися на більш легкі в реалізації і менш ресурсомісткі NetBEUI і IPX, то сучасні Windows орієнтуються на TCP/IP.

При використанні NetBEUI і IPX NetBIOS сам забезпечує надійність доставки даних (функціональність SPX не використовувати), а при використанні TCP/IP надійність доставки забезпечує TCP, за що удостоївся окремого імені «NBT».

Інтерфейс NetBIOS являє собою типовий інтерфейс взаємодії програм (API) для забезпечення мережевих операцій введення–виведення і управління транспортним протоколом.

Програми, що використовують NetBIOS API інтерфейс, можуть працювати тільки при наявності протоколу, що допускає використання такого інтерфейсу.

NetBIOS також визначає протокол, що функціонує на сеансовому/транспортному рівнях моделі OSI. Цей протокол використовується протоколами нижчих рівнів, такими як NBFP (NetBEUI) і NetBT для виконання

мережевих запитів вводу–виводу і операцій, описаних в стандартному интерфейсном наборі команд NetBIOS.

Тобто, NetBIOS сам не підтримує виконання файлових операцій. Ця функція покладається на протоколи нижчих рівнів, а сам NetBIOS забезпечує тільки зв'язок з цими протоколами і NetBIOS API–інтерфейс.

Пакет NETBIOS створений для використання групою ЕОМ. Підтримує як режим сесій (робота через з'єднання), так і режим дейтаграм (без встановлення з'єднання). 16–символьні імена об'єктів в NETBIOS розподіляються динамічно.

NETBIOS має власну DNS, яка може взаємодіяти з інтернетівський. Ім'я об'єкта при роботі з NETBIOS не може починатися з символу *.

Додатки можуть знайти через NETBIOS потрібні їм ресурси, встановити зв'язок і послати або отримати інформацію. NETBIOS використовує для служби імен порт 137, для служби дейтаграм – порт 138, а для сесій – порт 139. Будь–яка сесія починається з NETBIOS–запиту, завдання IP–адреси і визначення TCP–порту віддаленого об'єкта, далі йде обмін NETBIOS–повідомленнями, після чого сесія закривається.

Сесія здійснює обмін інформацією між двома NETBIOS – додатками. Довжина повідомлення лежить в межах від 0 до 131 071 байт. Припустимо одночасне встановлення декількох сесій між двома об'єктами.

При організації IP–транспорту через NETBIOS IP–дейтаграмма вкладається в NETBIOS–пакет. Інформаційний обмін відбувається в цьому випадку без встановлення зв'язку між об'єктами. Імена NETBIOS повинні містити в собі IP–адреси. Так, частина NETBIOS–адреси може мати вигляд IP.XX.XX. XX. XX, де IP вказує на тип операції (IP через Netbios), а XX. XX. XX.XX – IP– адреса.

Система NETBIOS має власну систему команд (call, listen, hang up, send, receive, session status, reset, cancel, adapter status, unlink, remote program load) і примітивів для роботи з дейтаграммами (send datagram, send broadcast datagram, receive datagram , receive broadcast datagram).

Всі кінцеві вузли NETBIOS діляться на три типи:

– широкомовні («b») вузли;

– вузли точка–точка («p»);

– вузли змішаного типу («m»).

IP–адреса може асоціюватися з одним із зазначених типів. B–вузли встановлюють зв'язок зі своїм партнером за допомогою широкомовних запитів. P– і M–вузли для цієї мети використовують netbios сервер імен (NBNS) і сервер розподілу дейтаграм (NBDD).

NetBIOS забезпечує:

– реєстрацію і перевірку мережевих імен;

– встановлення і розрив з'єднань;

– зв'язок з підтвердженням доставки інформації;

– зв'язок без підтвердження доставки інформації;

– підтримку управління і моніторингу драйвера і мережевої карти.

Peer-to-peer (рівний до рівного) – варіант архітектури системи, в основі якої стоїть мережа рівноправних вузлів.

Комп'ютерні мережі типу peer-to-peer (або P2P) засновані на принципі рівноправності учасників і характеризуються тим, що їх елементи можуть зв'язуватися між собою, на відміну від традиційної архітектури, коли лише окрема категорія учасників, яка називається серверами може надавати певні сервіси іншим.

Фраза «peer-to-peer» була вперше використана у 1984 році Парбауелом Йохнухуйтсманом (Parbawell Yohnuhuitsman) при розробці архітектури Advanced Peer to Peer Networking фірми IBM.

В чистій «peer-to-peer» мережі не існує поняття клієнтів або серверів, лише рівні вузли, які одночасно функціонують як клієнти та сервери по відношенню до інших вузлів мережі. Ця модель мережевої взаємодії відрізняється від клієнт-серверної архітектури, в якій зв'язок відбувається лише між клієнтами та центральним сервером.

Така організація дозволяє зберігати працездатність мережі при будь-якій конфігурації доступних її учасників. Проте практикується використання P2P мереж які все ж таки мають сервери, але їх роль полягає вже не у наданні сервісів, а у підтримці інформації з приводу сервісів, що надаються клієнтами мережі.

В P2P системі автономні вузли взаємодіють з іншими автономними вузлами. Вузли є автономними в тому сенсі, що не існує загальної влади, яка може контролювати їх. В результаті автономії вузлів, вони не можуть довіряти один одному та покладатися на поведінку інших вузлів, тому проблеми масштабування та надмірності стають важливішими ніж у випадку традиційної архітектури.

Сучасні P2P-мережі набули розвитку завдяки ідеям, пов'язаними з обміном інформацією, які формувалися у руслі того, кожен вузол може надавати та отримувати ресурси які надаються будь-якими іншими учасниками. У випадку мережі Napster, це був обмін музикою, в інших випадках це може бути надання процесорного часу для пошуку інопланетних цивілізацій (SETI@home) або ліків від раку (Folding@home).

Переваги P2P

Розподіл/зменшення вартості. Централізовані системи, які обслуговують багато клієнтів, зазвичай складають більшість вартості системи. Коли, ця вартість стає дуже великою, архітектура P2P може допомогти розподілити вартість серед користувачів. Наприклад, серед систем файлообміну Napster дозволив розподілити вартість зберігання файлів і міг підтримувати індекс, потрібний для сумісного використання. Економія коштів, здійснюється за допомогою використання та об'єднання ресурсів, які в іншому випадку не використовуються (наприклад SETI@home). Оскільки вузли зазвичай є автономними, важливо розподіляти витрати справедливо.

Об'єднання ресурсів. Децентралізований підхід веде до об'єднання ресурсів. Кожен вузол в системі P2P приносить певні ресурси як наприклад обчислювальна потужність або пам'ять. У програмах, які потребують величезну

кількість цих ресурсів, як наприклад intensive моделювання або розподілені файлові системи, природно використовувати P2P, щоб залучити ці ресурси. Розподілені обчислювальні системи, як наприклад SETI@Home, distributed.net, і Endeavours – очевидні приклади цього підходу. Об'єднуючи ресурси тисяч вузлів, вони можуть виконувати важкі з точки зору кількості обчислень функції. Файлобмінні системи, як наприклад Napster, Gnutella, і так далі, також об'єднують ресурси. У цих випадках, це дисковий простір, щоб зберігати дані, та пропускна спроможність, щоб їх передавати.

Вдосконалена масштабованість/надійність. З відсутністю сильної центральної влади по відношенню до автономних вузлів, важливою метою є покращення масштабованості і надійності. Масштабованість і надійність визначаються в традиційному для розподілених систем сенсі, як наприклад використання пропускної спроможності – скільки вузлів можуть бути досягнуті від одного вузла, скільки вузлів може підтримуватися, скільки користувачів може підтримуватися. Розподілена природа peer-to-peer мереж також збільшує помилкостійкість у разі невдач, шляхом дублювання даних поміж багатьох вузлів, і – в чистих системах P2P – надаючи можливість вузлу знайти дані без залежності від єдиного централізованого індексного сервера. У останньому випадку, немає ніякої єдиної критичної точки в системі.

Збільшена автономія. У багатьох випадках, користувачі розподіленої системи не бажають залежати від будь-якого централізованого постачальника послуг. Натомість, вони воліють, щоб всі дані та призначена для них робота виконувалась локально. Системи P2P підтримують цей рівень автономії, тому що вони вимагають, щоб кожен вузол робив необхідну для нього частину праці.

Анонімність/конфіденційність. Пов'язаним із автономією є поняття анонімності і конфіденційності. Користувач, можливо, не хоче, щоб кого-небудь або будь-який постачальник послуг знав про нього або про його роль у системі. З центральним сервером, гарантувати анонімність важко, тому що сервер зазвичай зможе ідентифікувати клієнта, як мінімум через його адресу в Інтернет.

Використовуючи структуру P2P, в якій дії виконуються локально, користувачі можуть уникати необхідності передавати будь-яку інформацію про себе до кого-небудь іншого. FreeNet – яскравий приклад того, як анонімність може вбудуватися в додаток P2P. Він пересилає повідомлення через інші вузли, щоб забезпечити неможливість вистежування початкового автора. Це збільшує анонімність, використовуючи ймовірнісні алгоритми таким чином, щоб походження не можливо було легко відстежити аналізуючи трафік у мережі.

Динамічність. Системи P2P припускають, що оточення надзвичайно динамічне. Тобто, ресурси, як наприклад вузли, з'являються та зникають із системи безперервно. У випадках комунікації, як наприклад мережі для обміну повідомленнями, використовуються так званий «список контактів», щоб інформувати користувачів, коли їхні друзі стають доступними. Без цього, потрібно було би, щоб користувачі «опитували» партнерів, посилаючи періодичні повідомлення. У випадку розподілених обчислень, як наприклад distributed.net і SETI@home, система повинна пристосуватись до заміни учасників. Тому вони повинні повторно видавати завдання для обчислення іншим учасникам, щоб гарантувати, що робота не втрачена, якщо попередні учасники відпадають від мережі, поки вони виконували крок обчислення.

Класифікація P2P систем

За функціями:

1. Розподілені обчислення. Обчислювальна проблема розподіляються на невеликі незалежні частини. Обробка кожної з частин робиться на індивідуальному ПК і результати збираються на центральному сервері. Цей центральний сервер відповідальний за розподілення елементів роботи серед окремих комп'ютерів в Інтернеті. Кожен із зареєстрованих користувачів має клієнтське програмне забезпечення. Воно користується періодами бездіяльності в ПК (часто це характеризується часами активації скрінсейверів), щоб виконувати деяке обчислення, надане сервером. Після того, як обчислення закінчене, результат посилається назад до сервера, і нова робота передається для клієнта.

2. Файлообмін. Зберігання та обмін даними – це одна з областей, де технологія P2P була найуспішнішою. Мультимедійні дані, наприклад, вимагають великих файлів. Napster і Gnutella використовувались користувачами, щоб обійти обмеження пропускної спроможності, які роблять передачу великих файлів неприйнятними.

3. Співпраця. Природа технології P2P робить її добре придатною для забезпечення співпраці між користувачами. Це може бути обмін повідомленнями, онлайн ігри, сумісна робота над документами в бізнесі, освіті та дома.

За ступенем централізації:

1. Чисті peer-to-peer системи. Вузли є рівними, поєднуючи ролі серверу та клієнту. Не існує центрального сервера, що керує мережею. Прикладами таких систем є Gnutella та Freenet.

2. Гібридні peer-to-peer системи. Мають центральний сервер, що зберігає інформацію про вузли та відповідає на запити відносно цієї інформації. Вузли займаються забезпеченням ресурсами (тому що центральний сервер їх не має), повідомленням сервера про наявність цих ресурсів надання ресурсів іншим вузлам, які бажають ними скористатися.

В залежності від того, як вузли з'єднуються один з одним можна поділити мережі на структуровані та неструктуровані:

1. Неструктурована мережа P2P формується, коли з'єднання встановлюються довільно. Такі мережі можуть бути легко сконструйовані, оскільки новий вузол, який хоче приєднатися до мережі, може скопіювати існуючі з'єднання іншого вузла, а вже потім почати формувати свої власні. У неструктурованій мережі P2P, якщо вузол бажає знайти певні дані в мережі, запит доведеться передати майже через всю мережу, щоб охопити так багато вузлів, як можливо. Головним недоліком таких мереж є те, що запити, можливо, не завжди вирішуються. Скоріш за все популярні дані будуть доступні в багатьох вузлів та пошук швидко знайде потрібне, але якщо вузол шукає рідкісні дані, наявні лише в декількох інших вузлів, то надзвичайно малоймовірно, що пошук буде успішним.

Оскільки немає ніякої кореляції між вузлами та даними, що вони зберігають, немає ніякої гарантії, що запит знайде вузол, який має бажані дані.

2. Структурована мережа P2P використовує єдиний алгоритм, щоб гарантувати, що будь-який вузол може ефективно передати запит іншому вузлу, який має бажаний файл, навіть якщо файл надзвичайно рідкісний. Така гарантія потребує структуровану систему з'єднань. У наш час найпопулярнішим типом структурованої мережі P2P є розподілені хеш-таблиці, в яких хешування використовується для встановлення зв'язку між даними та конкретним вузлом, який за них відповідає.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використати фіналіста конкурсу AES – шифр Rijndael. Він є нетрадиційним блоковим шифром, оскільки не використовує мережу Фейштеля для криптоперетворень. Алгоритм представляє кожний блок кодуємих даних у вигляді двовимірного масиву байт розміром 4x4, 4x6 або 4x8 залежно від установленої довжини блоку. Далі на відповідних етапах перетворення відбуваються або над незалежними стовпцями, або над незалежними рядками, або взагалі над окремими байтами в таблиці.

Всі перетворення в шифрі мають строге математичне обґрунтування. Сама структура й послідовність операцій дозволяють виконувати даний алгоритм ефективно як на 16-бітних так і на 64-бітних процесорах. У структурі алгоритму закладена можливість паралельного виконання деяких операцій, що на багатопроцесорних робочих станціях може ще підняти швидкість шифрування в 4 рази.

Алгоритм складається з деякої кількості раундів (від 10 до 14 – це залежить від розміру блоку й довжини ключа), у яких послідовно виконуються наступні операції :

ByteSub – Таблична підстановка 8x8 біт (рисунок 4.3).

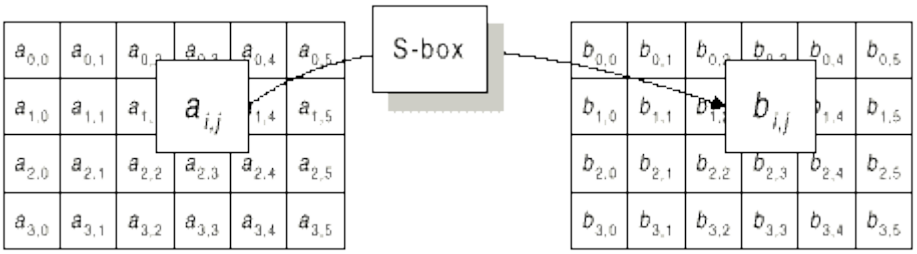

Рисунок 4.3 – Таблична підстановка 8x8 біт

ShiftRow – зрушення рядків у двовимірному масиві на різні зсуви (рисунок 4.4).

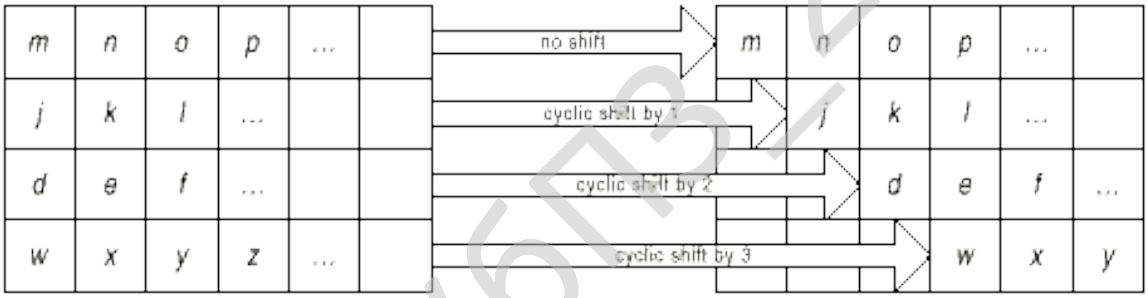

Рисунок 4.4 – Зрушення рядків у двовимірному масиві на різні зсуви

MixColumn – математичне перетворення, що перемішує дані усередині стовпця (рисунок 4.5).

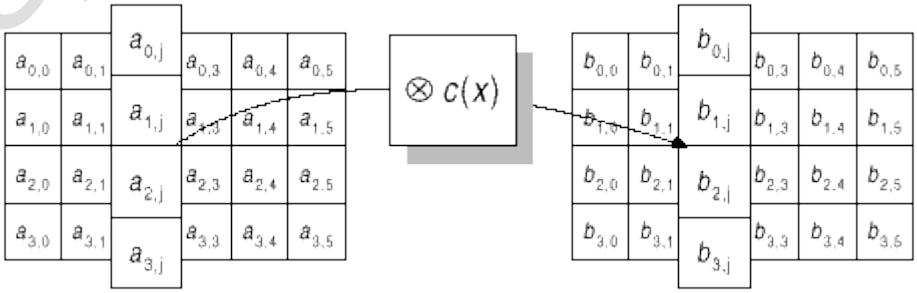

Рисунок 4.5 – Математичне перетворення, що перемішує дані усередині стовпця

AddRoundKey – додавання матеріалу ключа операцією XOR (рисунок 4.6).

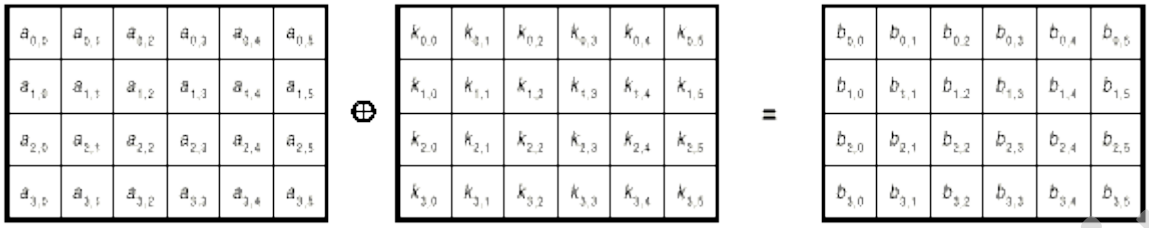

Рисунок 4.6 – Додавання матеріалу ключа операцією XOR

В останньому раунді операція перемішування стовпців відсутня, що робить всю послідовність операцій симетричною.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. Інтерфейс системи складається з наступних функціональних розділів:

– Функціональних кнопок ПЗ.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші: Налаштування; Довідка; Журнал роботи.

– Верхнього меню: Файл; Налаштування; Опції; Довідка.

– Розділу обрання групи.

– Розділу виведення результату роботи системи.

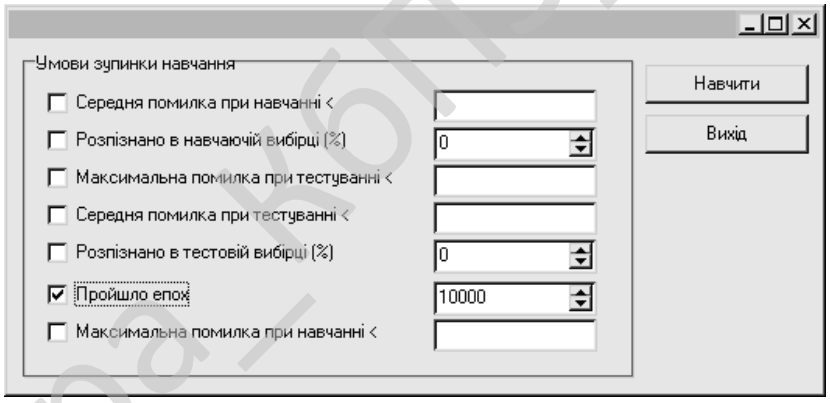

Рисунок 5.1 – Вікно введення даних ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких

помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

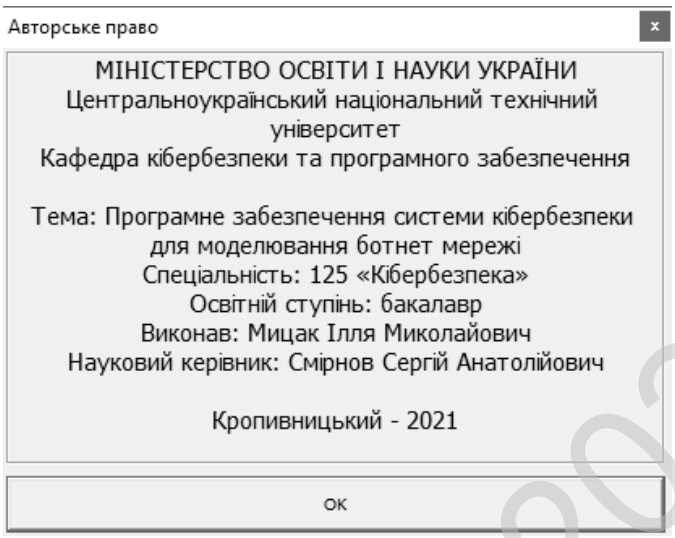

Рисунок 5.2 – Авторське право

Процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Таким чином у результаті вищерозглянутого можна стверджувати що розроблено інтерфейс системи у відповідності з вибраною метою роботи. Система містить максимальний необхідний набір функцій придатних для виконання будь-яких дій для забезпечення повноцінної роботи програми. Далі розглянемо висновки та використані літературні джерела.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи кібербезпеки для моделювання ботнет мережі.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем кібербезпеки для моделювання ботнет мережі.

– Досліджена система кібербезпеки для моделювання ботнет мережі.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для моделювання ботнет мережі.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання для моделювання ботнет мережі.

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 Delphi 10.4 Sydney. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки для моделювання ботнет мережі. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної

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

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм AES.

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. Бодянский Е.В. Нейро-фаззи сети Петри в задачах моделирования сложных систем. Монография (научное издание) / Е.В. Бодянский., Е.И. Кучеренко, А.И. Михалев – Дніпропетровськ: Системні технології, 2005. –311 с.

2. Вишневский В.М. Теоретические основы проектирования компьютерных сетей / В.М. Вишневский. – М.: Техносфера, 2003. – 512 с.

3. Конахович Г.Ф. Сети передачи пакетных данных / Г.Ф. Конахович, В.М.Чуприн. – К.:МК-Пресс, 2006. – 272 с.

4. Телекоммуникационные системы и сети: учебное пособие. В 3 томах / [В.В. Величко, Е.А. Субботин, В.П. Шувалов, А.Ф. Ярославцев]; под ред. В.П. Шувалова. – М.: Горячая линия-Телеком, 2005, т. 3 – 592 с.

5. Уолрэнд Дж. Телекоммуникационные и компьютерные сети / Дж. Уолрэнд. – М.: Постмаркет, 2001. – 480 с.

6. Ma Anwar. Integrating knowledge-base and Dijkstra's algorithm for finding best alternate route dynamically // Proceedings IEEE INMIC – 2003, P. 428-433.

7. Mohamed G. Gouda, Member, IEEE, and Marco Schneider. Maximizable Routing Metrics // IEEE/ACM Transactions on networking. – 2003. – №11(4), P. 663-675.

8. Joo Young Hwang, Jun Song Kim, Sang Seok Lim, and Kyu Ho Park. A Fast Path Planning by Path Graph Optimization // IEEE Transactions on systems, man, and cybernetics-part a: systems and humans. – 2003. – №1, P. 121-128.

9. Jui-Fa Chen, Wei-Chuan Lin. A Message Interchange Protocol based on Routing Information Protocol in a Virtual World / Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05) – 2005, P. 201-208.

10. Босько В.В. Анализ и сравнительное исследование перспективных направлений развития цифровых телекоммуникационных систем и сетей / Смирнов

А.А., Босько В.В., Мелешко Е.В. // Системи обробки інформації. – Х.: ХУ ПС, 2008. – Вип.7(74). – С.120-123.

11. Босько В.В. Разработка методики оценки середнего времени обслуживания информационных пакетов в телекоммуникационной сети / Смирнов А.А., Босько В.В., Мелешко Е.В. // Системи управління, навігації та зв'язку. – Київ: ДП «Центральний науково-дослідний інститут навігації і управління», 2009. – Вип. 2(10). – С.162-165.

12. Босько В.В. Разработка математической модели процесса динамического управления маршрутизацией данных в телекоммуникационной сети / Смирнов А.А., Босько В.В. // Системи управління, навігації та зв'язку. – К.: ДП «ЦНДІ навігації і управління», 2009. – Вип. 3(11). – С.208-210.

13. Босько В.В. Разработка метода определения оптимального множества путей передачи информации в телекоммуникационной сети / Смирнов А.А., Босько В.В. // Збірник наукових праць. – Х.: ХУПС, 2009. – Вип. 3(21). – С.102-108.

14. В.В. Босько. Разработка метода прогнозирования поведения информационного потока в телекоммуникационной сети // Збірник наукових праць. Харківського університету Повітряних Сил: – Х.:ХУ ПС, – 2010.-Вип. 3 (25) .- С.126-130.

15. Босько В.В. Исследование особенностей построения современных телекоммуникационных систем и сетей / Смирнов А.А., Босько В.В. // Проблемы информатики и моделирования. Материалы восьмой международной научно-технической конференции 26-28 ноября. – Х.: НТУ "ХПИ", 2008. – С.54.

16. Босько В.В. Исследование влияния времени мониторинга телекоммуникационной сети на точность прогнозирования поведения трафика / Смирнов А.А., Босько В.В. // Перша міжнародна науково-практична конференція "Проблеми й перспективи розвитку IT-індустрії" м. Харків , 18-19 листопада 2009р. – С.198-200.

17. Босько В.В. Анализ математических моделей телекоммуникационной сети / Смирнов А.А., Босько В.В. // Проблемы информатики и моделирования.

Материалы девятой международной научно-технической конференции 26-28 ноября. – Х.: НТУ “ХПИ”, 2009. – С.52-53.

18. Босько В.В. Метод повышения оперативности передачи данных в телекоммуникационной сети / Смирнов А.А., Босько В.В. // Збірник тез доповідей науково-практичної конференції “Застосування інформаційних технологій у підготовці та діяльності сил охорони правопорядку” 24-25 березня. – Х.: АВВ МВС України, 2010. – С.54.

19. Босько В.В. Динамическое управление процессом маршрутизации данных в телекоммуникационной сети / Смирнов А.А., Босько В.В. // Тези доповідей Міжнародної науково-практичної конференції м. Вінниця, Україна 19-21 травня 2010 року. – Вінниця: ВНТУ, 2010. – С.231-232.

20. Можаєв О.О. Часова прозорість мережі, як характеристика, що визначає виконання необхідної якості обслуговування / О.О. Можаєв, О.Д. Анохіна, С.Ю. Гайдаров, С.Г. Семенов // Системи обробки інформації. – Х.: ХВУ, 2004. – Вип. 11(39). – С.133-139.

21. Науменко М.І. Технології комп’ютерних мереж АСУ військами: підручник. / М.І. Науменко, Ю.В. Стасєв, І.В. Рубан – Х.: ХУ ПС, 2004. – 458 с.

22. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов / В.Г. Олифер, Н.А. Олифер. –2-е изд. – СПб.: Питер, 2007. – 958 с.

23. Семенов Ю.А. Сети Інтернет. Архитектура и протоколы / Ю.А. Семенов. – М.: Блик плюс, 1998. – 424 с.

24. Таненбаум Э. Компьютерные сети / Эндрю Таненбаум; пер. с англ. А. Леонтьев. – СПб.: Питер, 2002. — 848 с.

25. Чернявский Г.М. Новые технологии в спутниковых системах / Г.М. Чернявский // Информационные технологии и вычислительные системы. – М.: Институт микропроцессорных вычислительных систем РАН, 2005. – Вып.1 – С. 3 – 11.

26. Галкин В.А. Телекоммуникации и сети / В.А. Галкин,

Ю.А.Григорьев. – М.: МГТУ имени Н.Э. Баумана, 2003. – 608 с.

27. ДСТУ 2481 – 94 Системи оброблення інформації інтелектуальні інформаційні технології. Терміни та визначення. – Х.: ДЕРЖСТАНДАРТ УКРАЇНИ, 1994. – 33 с.

28. ДСТУ В 3265 – 95. Зв'язок військовий. Терміни та визначення. – К.:УкрНДІССІ, 1995. – 23 с.

29. ITU-T Y.1540. Internet protocol data communication service – IP packet transfer and availability performance parameters, 2007.

30. Пантелеев А.В. Т.А. Методы оптимизации в примерах и задачах / А.В. Пантелеев, Т.А. Летова. – М.: Высшая школа, 2002. – 544 с.

31. Свами М.Н., Тхуласираман К. Графы, сети и алгоритмы: пер. с англ. / М.Н. Свами, К. Тхуласираман; под ред.В.А. Горбатова. – М.: Мир, 1984. – 454 с.

32. An Chen, Albert Kai-Sun Wong, and Chin-Tau Lea, Senior Member. Routing and Time-Slot Assignment in Optical TDM Networks // IEEE Journal on selected areas in communications. – 2004. – №22(9), P. 1648-1657.

33. C. Casetti, R. Lo Cigno, M. Mellia, M. Munafo. A New Class of QoS Routing Strategies Based on Network Graph Reduction // Proceedings of IEEE INFOCOM. – 2002, P.715-722.

34. Chiara Francalanci and Paolo Giacomazzi. High-Performance Self-Routing Algorithm for Multiprocessor Systems with Shuffle Interconnections // IEEE Transactions on parallel and distributed systems. – 2006. – №1, P. 38-50.

35. T. D. Sudhakar, N. Shanmuga Vadivoo, S. Mary Raja Slochanal, S. Ravichandran. Supply Restoration In Distribution Networks Using Dijkstra's Algorithm / lnternatlonal Conference on Power System Technology – POWERCON 2004, Singapore, 21-24 November 2004, P. 640-645

36. Chris Loeser, Andre Brinkmann, Ulrich Ruckert. Distributed Path Selection (DPS) a Traffic Engineering Protocol for IP-Networks / Proceedings of the 37th Hawaii International Conference on System Sciences – 2004, P. 1-8.

37. Ji Li and Kwan L. Yeung. A Two-Step Approach to Restorable Dynamic

QoS Routing // IEEE Communications Society. – 2004, P. 1166-1170.

38. Jianbin Wei and Cheng-Zhong Xu. Feedback Control Approaches for Quality of Service Guarantees in Web Servers // NAFIPS 2005. Annual Meeting of the North American Fuzzy Information Processing Society – 2005. P. 700-705

39. Joao Luis Sobrinho. Algebra and Algorithms for QoS Path Computation and Hop-by-Hop Routing in the Internet // IEEE/ACM Transactions on networking. – 2002. – №10(4), P. 541-55.

40. Srihari Nelakuditi, Srivatsan Varadarajan, and Zhi-Li Zhang. On Localized Control in QoS Routing // IEEE Transactions on Automatic Control. – 2002. – №47(6), P. 1026-1032.

41. Song Gao, Ismail Chabini. Policy-Based Stochastic Dynamic Traffic Assignment Models And Algorithms / The IEEE 5th international conference on intelligent transportation system, 3-6 september 2002, Singapore, P. 445-453.

42. Stefan Soucek, Member, IEEE, and Thilo Sauter, Member. Quality of Service Concerns in IP-Based Control Systems // IEEE Transactions on industrial electronics. – 2004. – №51(6), P. 1249-1258.

43. Ji Li and Kwan L. Yeung. A Two-Step Approach to Restorable Dynamic QoS Routing // IEEE Communications Society. – 2004, P. 1166-1170.

44. Sudha Krishnamurthy, William H. Sanders, Fellow, IEEE, and Michel Cukier, Member. An Adaptive Quality of Service Aware Middleware for Replicated Services //IEEE Transactions on parallel and distributed systems. – 2003. – №14(11), P. 1112-1125.

45. Xin Yuan , Xingming Liu. Heuristic Algorithms for Multi–Constrained Quality of Service Routing // Proceedings of IEEE INFOCOM. – 2001, P. 844-853.

46. Anees Shaikh, Jennifer Rexford, and Kang G. Shin. Evaluating the Impact of Stale Link State on Quality-of-Service Routing // IEEE/ACM Transactions on Networking. – 2001. – №9(2), P. 162-176.

47. Семенов С.Г. Разработка распределенного метода многопутевой маршрутизации, основанного на потоковой модели с предвычислением путей

(маршрутов) / С.Г. Семенов, А.Г. Беленков, А.А. Можаев // Моделювання та інформаційні технології. – К.: ІПМЕ ім. Г.Є.Пухова, – 2005. – Вип. 32. – С.189-192.

48. Галушкин А.И. Теория нейронных сетей. Уч. пособие для вузов/ Александр Иванович Галушкин – М.: ИПРЖР, 2000. – 416 с.

49. Конахович Г.Ф. Сети передачи пакетных данных / Г.Ф. Конахович, В.М.Чуприн. – К.:МК-Пресс, 2006. – 272 с.

50. Гмурман В.Е. Теория вероятностей и математическая статистика / Владимир Ефимович Гмурман. – М.: Высшая школа, 2003. – 479 с.

51. Кульгин М.Б. Технология и маршрутизация IP/IPX трафика / Максим Кульгин. – М.: Компьютер-пресс, 1998. – 324 с.

52. Лемешко А.В. Теоретические основы управления сетевыми ресурсами с использованием тензорных математических моделей телекоммуникационных систем: Дисс… доктора Техн. наук: 05.12.02. – Х., 2006. – 347 с.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для моделювання ботнет мережі.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 185-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи кібербезпеки для моделювання ботнет мережі.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

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

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для моделювання ботнет мережі;

– 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 архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 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 XP/Vista/7/8/10.

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 Delphi 10.4 Sydney.

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 – 71 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

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.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 7.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки для моделювання ботнет мереж системи кібербезпеки для моделювання ботнет мережії

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 64

Літера: РП

Кропивницький – 2021 року

Основна програма

BotnetNetwork.pas – ініціалізація базових констант

```
unit BotnetNetwork;

interface

const

  DefaultAlpha = 1;              // Параметр активаційної функції
  DefaultEpochCount = 10000;     // Кількість етапів для навчання
  DefaultErrorValue = 0.05;      // Помилка за замовчуванням для всіх видів
  DefaultHopfLayerCount = 2;     // Кількість шарів у мереж системи кібербезпеки
для моделювання ботнет мережіі Хопфилда
  DefaultLayerCount = 0;         // Мінімальна кількість шарів у мереж системи
кібербезпеки для моделювання ботнет мережіі back-propagation
  DefaultMaxIterCount = 10;      // Максимальна кількість ітерацій в алгоритмі
Хопфилда
  DefaultMomentum = 0.9;         // Імпульс - момент
  DefaultNeuronCount = 0;        // Кількість нейронів у прихованому шарі за
замовчуванням
  DefaultPatternCount = 0;       // Кількість прикладів
  DefaultTeachRate = 0.1;        // Швидкість навчання
  DefaultTeachIdentCount = 100;  // Необхідний відсоток розпізнаних прикладів з
навчальної множини
  DefaultTestIdentCount = 100;   // Необхідний відсоток розпізнаних прикладів з
тестової множини
  DefaultUseForTeach = 100;      // Відсоток прикладів використовуваних як
навчальна множина
  DefaultDeltaBarAcceleratingConst = 0.095;
  DefaultDeltaBarDecFactor = 0.85;
  DefaultRPropInitValue = 0.1;
  DefaultRPropMaxStepSize = 50;
  DefaultRPropMinStepSize = 1 E-6;
  DefaultRPropDecFactor = 0.5;
  DefaultRPropIncFactor = 1.2;
  DefaultSuperSABDecFactor = 0.5;
  DefaultSuperSABIncFactor = 1.05;

  SensorLayer = 0;               // Сенсорний шар
  BiasNeuron = 1;                // Зсув у мереж системи кібербезпеки для
моделювання ботнет мережіі back-propagation

  Separators = ['.', ','];
  Letters    = ['a'..'z'];
  Capitals   = ['A'..'Z'];
  DigitChars = ['0'..'9'];
  SpaceChar = #9;

resourcestring

  SFieldNorm = 'Не існує типу нормалізації %d';
  SFieldKind = 'Не існує типу поля %d';
  SFieldIndexRange = 'Неправильно зазначений номер поля %d';
  SInFieldCount = 'Неправильно встановлена кількість вхідних полів';
  SInNeuronCount = 'Неправильно встановлена кількість вхідних нейронів';
  SInVectorCount = 'Неправильно встановлена розмірність вхідного вектора';
  SLayerRangeIndex = 'Неправильно зазначений номер шару %d';
  SNeuronRangeIndex = 'Неправильно зазначений номер нейрона %d';
  SNeuronCount = 'Неправильно зазначена кількість нейронів';
  SOutFieldCount = 'Неправильно встановлена кількість вихідних полів';
  SOutNeuronCount = 'Неправильно встановлена кількість вихідних нейронів';
  SOutVectorCount = 'Неправильно встановлена розмірність вихідного вектора';
  SPatternRangeIndex = 'Вихід за межі масиву прикладів %d';
```

```
  SStreamCannotRead = 'Помилка читання з потоку';
  SWeightRangeIndex = 'Неправильно зазначений номер ваги %d';
  SWrongFileName = 'Неправильно зазначене ім'я файлу %s';
  SCannotBeNumber = 'Помилка, вираз %s неможливо привести до числового типу';
  SBPStopCondition = 'Не задана умова зупинки процесу навчання';

type

  TVectorInt = array of integer;
  TVectorFloat = array of double;
  TVectorString = array of string;
  TMatrixInt = array of array of integer;
  TMatrixFloat = array of array of double;
  TNormalize = (nrmLinear, nrmSigmoid, nrmAuto,  nrmNone,
                nrmLinearOut, nrmAutoOut);
  TNeuroFieldType = (fdInput, fdOutput, fdNone);

implementation


end.
```

BotnetNetworkEditor.pas – проектування нейронних мереж системи кібербезпеки для моделювання ботнет мережі

```
unit BotnetNetworkEditor;

interface

uses
  Classes,
  DesignIntf, DesignEditors,
  BotnetNetworkComp, BotnetNetworkEditorForm,
  SysUtils, Dialogs, Controls, BotnetNetworkEditorFieldsForm;

type
  TNeuronsInLayerFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

  TFileNameFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure Edit; override;
  end;

  TOptionsFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

procedure Register;

implementation

function TOptionsFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TOptionsFieldsProperty.Edit;
var
  i: integer;
  xNeuralNetExtended: TNeuralNetExtended;
  frmNeuroFields: TfrmNeuroFields;
  xCurrent: integer;
begin
  frmNeuroFields := TfrmNeuroFields.Create(nil);
  try
    with frmNeuroFields do
    begin
      xNeuralNetExtended := (GetComponent(0) as TNeuralNetExtended);
      for i := 0 to xNeuralNetExtended.AvailableFieldsCount - 1 do
        ltbFieldName.Items.Add(xNeuralNetExtended.Fields[i].Name);
      xCurrent := 0;
      rdgFieldType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].Kind;
      rdgNormType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].NormType;
      edtAlpha.Text := FloatToStr(xNeuralNetExtended.Fields[xCurrent].Alpha);
      sttMin.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMin);
      sttMax.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMax);
      NeuralNetExtended := xNeuralNetExtended;
      ShowModal;
```

```
    end;
  except
    frmNeuroFields.Free;
  end;
end;

function TOptionsFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetExtended) do
    if AvailableFieldsCount > 1 then
    begin
      for i := 0 to AvailableFieldsCount - 1 do
        Result := Result + Fields[i].Name + ',';
      Result[Length(Result)] := ']'
    end
    else
      Result[Length(Result) + 1] := ']';
end;

function TNeuronsInLayerFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

function TNeuronsInLayerFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetBP) do
    if LayerCount > 0 then
    begin
      for i := 0 to LayerCount - 1 do
        Result := Result + IntToStr(LayersBP[i].NeuronCount) + ',';
      Result[Length(Result)] := ']';
    end
    else
      Result[Length(Result) + 1] := ']';
end;

procedure TNeuronsInLayerFieldsProperty.Edit;
var
  i: integer;
  xPreviousCount: integer;
  xNeuralNetBP: TNeuralNetBP;
  frmNeuronsInLayer: TfrmNeuronsInLayer;
  xChangesMade: boolean;
begin
  xChangesMade := False;
  frmNeuronsInLayer := TfrmNeuronsInLayer.Create(nil);
  try
    with frmNeuronsInLayer do
    begin
      xNeuralNetBP := (GetComponent(0) as TNeuralNetBP);
      speLayers.Value := xNeuralNetBP.LayerCount;
      stgNeuronsInLayer.RowCount := xNeuralNetBP.LayerCount + 1;
      for i := 0 to xNeuralNetBP.LayerCount - 1 do
      begin
        stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
        stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(xNeuralNetBP.LayersBP[i].NeuronCount);
      end;
      if ShowModal = mrOk then
      begin
```

```
        xNeuralNetBP.ResetLayers;
        if speLayers.Value = 0 then
        begin
          xNeuralNetBP.ResetLayers;
          Exit;
        end;
        if xNeuralNetBP.LayerCount <> speLayers.Value then
          xChangesMade := True
        else
          for i := 0 to xNeuralNetBP.LayerCount - 1 do
            if xNeuralNetBP.LayersBP[i].NeuronCount <>
StrToInt(stgNeuronsInLayer.Cells[1, i + 1]) then
            begin
              xChangesMade := True;
              Break;
            end;
        if xChangesMade then
        begin
          if xNeuralNetBP.LayerCount = speLayers.Value then
            for i := 0 to speLayers.Value - 1 do
              xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1]
          else
            if xNeuralNetBP.LayerCount < speLayers.Value then
            begin
              for i := 0 to xNeuralNetBP.LayerCount - 1 do
                xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1];
              for i := xNeuralNetBP.LayerCount to speLayers.Value - 1 do
                xNeuralNetBP.AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i +
1]));
            end
            else
            begin
              if speLayers.Value > 0 then
                for i := 0 to speLayers.Value - 1 do
                  xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i
+ 1];
              xPreviousCount := xNeuralNetBP.LayerCount -  speLayers.Value;
              for i := 1 to xPreviousCount do
                xNeuralNetBP.DeleteLayer(xNeuralNetBP.LayerCount - 1);
            end;
          if not xNeuralNetBP.AutoInit then
            xNeuralNetBP.AutoInit := True;
        end;
      end;
    end;
  except
    frmNeuronsInLayer.Free;
  end;
end;

function TFileNameFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TFileNameFieldsProperty.Edit;
var
  xOpenDialog: TOpenDialog;
begin
  xOpenDialog := TOpenDialog.Create(nil);
  if UpperCase(GetName) = 'FILENAME' then
    xOpenDialog.Filter := 'Нейронна мереж системи кібербезпеки для моделювання
ботнет мережіа(*.nnw)|*.nnw|всі файли (*.*)|*.*';
  if UpperCase(GetName) = 'SOURCEFILENAME' then
    xOpenDialog.Filter := 'Текстові файли (*.txt)|*.txt|всі файли (*.*)|*.*';

  if xOpenDialog.Execute then
```

```
  begin
    if UpperCase(GetName) = 'FILENAME' then
      (GetComponent(0) as TNeuralNetExtended).FileName := xOpenDialog.FileName;
    if UpperCase(GetName) = 'SOURCEFILENAME' then
      (GetComponent(0) as TNeuralNetExtended).SourceFileName :=
xOpenDialog.FileName;
  end;
  xOpenDialog.Free;
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(TStrings), TNeuralNetBP, 'NeuronsInLayer',
TNeuronsInLayerFieldsProperty);
  RegisterPropertyEditor(TypeInfo(TFileName), TNeuralNetExtended, '',
TFileNameFieldsProperty);
  RegisterPropertyEditor(TypeInfo(string), TNeuralNetExtended, 'Options',
TOptionsFieldsProperty);
end;

end.
```

BotnetNetworkEditorForm.pas – редактор нейронних мереж системи кібербезпеки для моделювання ботнет мережі

```
unit BotnetNetworkEditorForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, StdCtrls, ExtCtrls, BotnetNetworkComp, Spin, BotnetNetwork;

type
  TfrmNeuronsInLayer = class(TForm)
    Bevel1: TBevel;
    btnOk: TButton;
    btnCancel: TButton;
    stgNeuronsInLayer: TStringGrid;
    Label1: TLabel;
    speLayers: TSpinEdit;
    procedure stgNeuronsInLayerGetEditMask(Sender: TObject; ACol,
      ARow: Integer; var Value: String);
    procedure stgNeuronsInLayerSetEditText(Sender: TObject; ACol,
      ARow: Integer; const Value: String);
    procedure speLayersChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmNeuronsInLayer: TfrmNeuronsInLayer;

implementation

{$R *.DFM}

procedure TfrmNeuronsInLayer.stgNeuronsInLayerGetEditMask(Sender: TObject;
  ACol, ARow: Integer; var Value: String);
begin
  Value := '0000';
end;

procedure TfrmNeuronsInLayer.stgNeuronsInLayerSetEditText(Sender: TObject;
  ACol, ARow: Integer; const Value: String);
begin
{  with stgNeuronsInLayer do
    try
      Cells[ACol, ARow] := IntToStr(StrToInt(trim(Value)));
    except
      Cells[ACol, ARow] := IntToStr(DefaultNeuronCount);
    end;}
end;

procedure TfrmNeuronsInLayer.speLayersChange(Sender: TObject);
var
  i: integer;
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  for i := 1 to stgNeuronsInLayer.RowCount do
    if trim(stgNeuronsInLayer.Cells[1, i]) = '' then
    begin
      stgNeuronsInLayer.Cells[0, i] := IntToStr(i);
      stgNeuronsInLayer.Cells[1, i] := IntToStr(DefaultNeuronCount);
    end;
end;
```

```
procedure TfrmNeuronsInLayer.FormCreate(Sender: TObject);
begin
  stgNeuronsInLayer.Cells[0,0] := '# шару';
  stgNeuronsInLayer.Cells[1,0] := 'нейронів';
end;

end.
```

BotnetNetworkEditorFieldsForm.pas – редактор властивостей полів нейроної мереж системи кібербезпеки для моделювання ботнет мережіі

```
unit BotnetNetworkEditorFieldsForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, BotnetNetworkComp;

type
  TfrmNeuroFields = class(TForm)
    ltbFieldName: TListBox;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    sttMin: TStaticText;
    sttMax: TStaticText;
    edtAlpha: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    Bevel2: TBevel;
    Label4: TLabel;
    procedure ltbFieldNameClick(Sender: TObject);
    procedure rdgFieldTypeClick(Sender: TObject);
    procedure rdgNormTypeClick(Sender: TObject);
    procedure edtAlphaChange(Sender: TObject);
  private
    { Private declarations }
  public
    NeuralNetExtended: TNeuralNetExtended;
    { Public declarations }
  end;

var
  frmNeuroFields: TfrmNeuroFields;

implementation

{$R *.DFM}

procedure TfrmNeuroFields.ltbFieldNameClick(Sender: TObject);
begin
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  edtAlpha.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  sttMin.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  sttMax.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

procedure TfrmNeuroFields.rdgFieldTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
```

```
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].Kind := rdgFieldType.ItemIndex;
end;

procedure TfrmNeuroFields.rdgNormTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].NormType := rdgNormType.ItemIndex;
end;

procedure TfrmNeuroFields.edtAlphaChange(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha :=
StrToFloat(edtAlpha.Text);
end;

end.
```

BotnetNetworkComp.pas – формування бібліотеки шаблонів та дослідження нейронних мереж системи кібербезпеки для моделювання ботнет мережі

```
unit BotnetNetworkComp;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, PumpData, BotnetNetwork, IniFiles, Math;

type

  // Класи виключень
  EInOutDimensionError = class(Exception);
  ENeuronCountError  = class(Exception);
  ENeuronNotEqualFieldError  = class(Exception);
  EBPStopCondition  = class(Exception);

  // Процедурні типи
  TActivation = function (Value: double): double of object;

  // Упереджуюче оголошення класів
  TNeuron = class;
  TLayer = class;

  // Базовий клас нейрона

  TNeuron = class(TObject)
  private
    FOutput: double;
    // Вектор ваг
    FWeights: TVectorFloat;
    // Покажчик на шар, у якому перебуває нейрон
    Layer: TLayer;
    function GetWeights(Index: integer): double;
    procedure SetWeights(Index: integer; Value: double);
    procedure SetWeightCount(const Value: integer);
  public
    constructor Create(ALayer: TLayer); virtual;
    destructor Destroy; override;
    // Ініціалізація ваг
    procedure InitWeights;  virtual;
    // Зважена сума
    procedure ComputeOut(const AInputs: TVectorFloat); virtual;
    property Output: double read FOutput write FOutput;
    property WeightCount: integer  write SetWeightCount;
    property Weights[Index: integer]: double read GetWeights write SetWeights;
  end;

  // Клас нейрона для мереж системи кібербезпеки для моделювання ботнет мережіі
Хопфілда

  TNeuronHopf = class(TNeuron)
  public
    procedure ComputeOut(const AInputs: TVectorFloat); override;
  end;

  // Клас нейрона для мереж системи кібербезпеки для моделювання ботнет мережіі

  TNeuronBP = class(TNeuron)
  private
    // Локальна помилка
    FDelta: double;
    // Значення швидкості навчання на попередньому етапі
    FLearningRate: TVectorFloat;
    // Значення частинної похідної на попередньому етапі
```

```
    FPrevDerivative: TVectorFloat;
    // Значення корекції ваги на попередньому етапі
    FPrevUpdate: TVectorFloat;
    // Функція активації
    FOnActivation: TActivation;
    // Похідна функції активації
    FOnActivation: TActivation;
    function GetPrevUpdate(Index: integer): double;
    function GetPrevDerivative(Index: integer): double;
    function GetLearningRate(Index: integer): double;
    function GetPrevUpdateCount: integer;
    procedure SetPrevDerivative(Index: integer; const Value: double);
    procedure SetPrevDerivativeCount(const Value: integer);
    procedure SetDelta(Value: double);
    procedure SetPrevUpdate(Index: integer; Value: double);
    procedure SetPrevUpdateCount(const Value: integer);
    procedure SetLearningRate(Index: integer; const Value: double);
    procedure SetLearningRateCount(const Value: integer);
  public
    destructor Destroy; override;
    procedure ComputeOut(const AInputs: TVectorFloat); override;
    property Delta: double read FDelta write SetDelta;
    property LearningRate[Index: integer]: double read GetLearningRate write
SetLearningRate; //
    property LearningRateCount: integer write SetLearningRateCount;
    property PrevDerivativeCount: integer write SetPrevDerivativeCount;
    property PrevDerivative[Index: integer]: double read GetPrevDerivative write
SetPrevDerivative; //
    property PrevUpdateCount: integer read GetPrevUpdateCount write
SetPrevUpdateCount;
    property PrevUpdate[Index: integer]: double read GetPrevUpdate write
SetPrevUpdate;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
  end;

  // Базовий клас шару
  TLayer = class(TPersistent)
  private
    FNumber: integer;
    // Розмірність NeuronCount
    FNeurons: array of TNeuron;
    function GetNeurons(Index: integer): TNeuron;
    function GetNeuronCount: integer;
    procedure SetNeurons(Index: integer; Value: TNeuron);
    procedure SetNeuronCount(Value: integer);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); virtual;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property Neurons[Index: integer]: TNeuron read GetNeurons write SetNeurons;
    property NeuronCount: integer read GetNeuronCount write SetNeuronCount;
  end;

  // Клас шару для мереж системи кібербезпеки для моделювання ботнет мережіі
Хопфілда
  TLayerHopf = class(TLayer)
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
  end;

  // Клас шару для мереж системи кібербезпеки для моделювання ботнет мережіі
  TLayerBP = class(TLayer)
  private
    function GetNeuronsBP(Index: integer): TNeuronBP;
    procedure SetNeuronsBP(Index: integer; Value: TNeuronBP);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
    destructor Destroy; override;
```

```
    procedure Assign(Source: TPersistent); override;
    property NeuronsBP[Index: integer]: TNeuronBP read GetNeuronsBP write
SetNeuronsBP;
  end;

  // Базовий клас мереж системи кібербезпеки для моделювання ботнет мережіі
  TNeuralNet = class(TComponent)
  private
    // Масив шарів
    FLayers: array of TLayer;
    // Число вибірок
    FPatternCount: integer;
    // Розмірність FPatternCount, InputNeuronCount
    FPatternsInput: TMatrixFloat;
    // Розмірність FPatternCount, OutputNeuronCount
    FPatternsOutput: TMatrixFloat;
    function GetLayers(Index: integer): TLayer;
    function GetOutputNeuronCount: integer;
    function GetPatternsOutput(PatternIndex: integer; OutputIndex: integer):
double;
    function GetPatternsInput(PatternIndex: integer; InputIndex: integer):
double;
    procedure SetLayers(Index: integer; Value: TLayer);
    procedure SetPatternsInput(PatternIndex: integer; InputIndex: integer;
Value: double);
    procedure SetPatternsOutput(PatternIndex: integer; InputIndex: integer;
Value: double);
  protected
    function GetLayerCount: integer; virtual;
    function GetInputNeuronCount: integer; virtual;
    procedure Clear; virtual;
    procedure ResizeInputDim; virtual;
    procedure ResizeOutputDim; virtual;
    procedure SetPatternCount(const Value: integer); virtual;
    procedure SetLayerCount(Value: integer); virtual;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  public
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); virtual; abstract;
    procedure AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat); overload; virtual;
    procedure DeleteLayer(Index: integer); virtual; abstract;
    procedure DeletePattern(Index: integer); virtual;
    procedure Init(const ANeuronsInLayer: TVectorInt); overload; virtual;
    property InputNeuronCount: integer read GetInputNeuronCount;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Layers[Index: integer]: TLayer read GetLayers write SetLayers;
    property OutputNeuronCount: integer read GetOutputNeuronCount;
    property PatternsInput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsInput write SetPatternsInput;
    property PatternsOutput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsOutput write SetPatternsOutput;
    procedure ResetPatterns; virtual;
  end;

  // Клас мереж системи кібербезпеки для моделювання ботнет мережіі Хопфілда
  TNeuralNetHopf = class(TNeuralNet)
  private
    FAutoInit: boolean;
    FInputNeuronCount: integer;
    FMaxIterCount: integer;
    FPatternCount: integer;
    FPatterns: TMatrixInt;
    FOnAfterInit: TNotifyEvent;
    FOnBeforeInit: TNotifyEvent;
    FOnPatternRecognized: TNotifyEvent;
    function GetInput(Index: integer): double;
    function GetPatterns(InputIndex: integer; PatternIndex: integer): integer;
    function Stabled: boolean;
```

```
    procedure SetInput(Index: integer; Value: double);
    procedure SetPatterns(InputIndex: integer; PatternIndex: integer; Value:
integer);
  protected
    function GetInputNeuronCount: integer; override;
    function GetLayerCount: integer; override;
    procedure SetInputNeuronCount(Value: integer);
    procedure SetPatternCount(const Value: integer); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddPattern(const ANewPattern: TVectorInt); reintroduce; overload;
    procedure Calc; virtual;
    procedure DeletePattern(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure InitWeights; virtual;
    procedure ResetPatterns; override;
    procedure ResizePatternsDim; virtual;
    property Input[Index: integer]: double read GetInput write SetInput;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Patterns[InputIndex: integer; PatternIndex: integer]: integer read
GetPatterns write SetPatterns;
  published
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property InputNeuronCount: integer read GetInputNeuronCount write
SetInputNeuronCount;
    property MaxIterCount: integer read FMaxIterCount write FMaxIterCount;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnPatternRecognized: TNotifyEvent read FOnPatternRecognized write
FOnPatternRecognized;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  end;

  // Клас мереж системи кібербезпеки для моделювання ботнет мережіі
  TNeuralNetBP = class(TNeuralNet)
  private
    // Коефіцієнт крутості граничної сигмоїдальної функції
    FAlpha: double;
    // Прапор автоініціалізації топології мереж системи кібербезпеки для
моделювання ботнет мережіі
    FAutoInit: boolean;
    // Прапор продовження навчання
    FContinueTeach: boolean;
    // Бажаний вихід нейромереж системи кібербезпеки для моделювання ботнет
мережіі розмірність OutputNeuronCount
    FDesiredOut: TVectorFloat;
    // Прапор зупинки при досягненні FEpochCount
    FEpoch: boolean;
    // Лічильник етапів (пред'явлення мереж системи кібербезпеки для моделювання
ботнет мережіі всіх прикладів з навчальної вибірки)
    FEpochCount: integer;
    // Номер поточної епохи
    FEpochCurrent: integer;
    // Значення помилки, при якій приклад вважається розпізнаним
    FIdentError: double;
    // Значення максимальної помилки на навчальній множині
    FMaxTeachResidual: double;
    // Значення максимальної помилки на тестовій множині
    FMaxTestResidual: double;
    // Значення середньої помилки на навчальній множині
    FMidTeachResidual: double;
    // Значення середньої помилки на тестовій множині
    FMidTestResidual: double;
    // Помилка на навчальній множині
    FTeachError: double;
    // Коефіцієнт інерційності
    FMomentum: double;
    // Кількість нейронів у шарах
```

```
    FNeuronsInLayer: TStrings;
    // Подія після ініціалізації
    FOnAfterInit: TNotifyEvent;
    FOnAfterNeuronCreated: TNotifyEvent;
    // Подія після навчання
    FOnAfterTeach: TNotifyEvent;
    // Подія до ініціалізації
    FOnBeforeInit: TNotifyEvent;
    // Подія до початку навчання
    FOnBeforeTeach: TNotifyEvent;
    // Подія після проходження одного етапу
    FOnEpochPassed: TNotifyEvent;
    // Число прикладів у навчальній безлічі
    FPatternCount: integer;
    // Масив утримуючий псевдовипадкову послідовність
    FRandomOrder: TVectorInt;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTeachCount: integer;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTestCount: integer;
    // Прапор зупинки навчання
    FStopTeach: boolean;
    FTeachStopped: boolean;
    // Коефіцієнт швидкості навчання - величина градієнтного кроку
    FTeachRate: double;
    // Число прикладів у тестовій множині
    FTestSetPatternCount: integer;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatterns: TMatrixFloat;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatternsOut: TMatrixFloat;
    function GetDesiredOut(Index: integer): double;
    function GetLayersBP(Index: integer): TLayerBP;
    function GetTestSetPatterns(InputIndex, PatternIndex: integer): double;
    function GetTestSetPatternsOut(InputIndex, PatternIndex: integer): double;
    procedure NeuronCountError;
    procedure NeuronsInLayerChange(Sender: TObject);
    procedure SetAlpha(Value: double);
    procedure SetDesiredOut(Index: integer; Value: double);
    procedure SetEpochCount(Value: integer);
    procedure SetLayersBP(Index: integer; Value: TLayerBP);
    procedure SetMomentum(Value: double);
    procedure SetTeachRate(Value: double);
    procedure SetTestSetPatternCount(const Value: integer);
    procedure SetTestSetPatterns(InputIndex, PatternIndex: integer; const Value:
double);
    procedure SetTestSetPatternsOut(InputIndex, PatternIndex: integer; const
Value: double);
    // Перетасування набору даних
    procedure Shuffle;
  protected
    function GetLayerCount: integer; override;
    function GetOutput(Index: integer): double; virtual;
    // Активаційна функція
    function Activation(Value: double): double; virtual;
    // Похідна активаційної функції
    function Activation(Value: double): double; virtual;
    // Середня квадратична помилка
    function QuadError: double; virtual;
    // Підстроювання ваг
    procedure AdjustWeights; virtual;
    // Розраховує  локальну помилку - дельту
    procedure CalcLocalError; virtual;
    // Перевірка мереж системи кібербезпеки для моделювання ботнет мережіі на
тестовій множині
    procedure CheckTestSet; virtual;
    procedure DoOnAfterInit; virtual;
    procedure DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex: integer);
virtual;
```

```
    procedure DoOnAfterTeach; virtual;
    procedure DoOnBeforeInit; virtual;
    procedure DoOnBeforeTeach; virtual;
    procedure DoOnEpochPassed; virtual;
    // Ініціалізація ваг мереж системи кібербезпеки для моделювання ботнет
мережіі псевдовипадковими значеннями
    procedure InitWeights; virtual;
    // Пред'явлення мереж системи кібербезпеки для моделювання ботнет мережіі
вхідних значень приклада
    procedure LoadPatternsInput(APatternIndex :integer); virtual;
    // Пред'явлення мереж системи кібербезпеки для моделювання ботнет мережіі
вхідних значень приклада
    procedure LoadPatternsOutput(APatternIndex :integer); virtual;
    // Поширює сигнал у прямому напрямку
    procedure Propagate; virtual;
    // Установка значень за замовчуванням
    procedure SetDefaultProperties; virtual;
    procedure SetPatternCount(const Value: integer); override;
    // Струс мереж системи кібербезпеки для моделювання ботнет мережіі
    procedure ShakeUp; virtual;
    property TeachStopped: boolean read FTeachStopped write FTeachStopped;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); override;
    procedure Compute(AVector: TVectorFloat); virtual;
    procedure DeleteLayer(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure ResetLayers; virtual;
    procedure TeachOffLine; virtual;
    property DesiredOut[Index: integer]: double read GetDesiredOut write
SetDesiredOut;
    property EpochCurrent: integer read FEpochCurrent;
    property IdentError: double read FIdentError write FIdentError;
    property LayersBP[Index: integer]: TLayerBP read GetLayersBP write
SetLayersBP;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Output[Index: integer]: double read GetOutput;
    property StopTeach: boolean read FStopTeach write FStopTeach;
    property TeachError: double read FTeachError;
    property MaxTeachResidual: double read FMaxTeachResidual;
    property MaxTestResidual: double read FMaxTestResidual;
    property MidTeachResidual: double read FMidTeachResidual;
    property MidTestResidual: double read FMidTestResidual;
    property RecognizedTeachCount: integer read FRecognizedTeachCount;
    property RecognizedTestCount: integer read FRecognizedTestCount;
    property TestSetPatternCount: integer read FTestSetPatternCount write
SetTestSetPatternCount;
    property TestSetPatterns[InputIndex: integer; PatternIndex: integer]: double
read GetTestSetPatterns write SetTestSetPatterns;
    property TestSetPatternsOut[InputIndex: integer; PatternIndex: integer]:
double read GetTestSetPatternsOut write SetTestSetPatternsOut;
  published
    property Alpha: double read FAlpha write SetAlpha;
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property ContinueTeach: boolean read FContinueTeach write FContinueTeach;
    property Epoch: boolean read FEpoch write FEpoch;
    property EpochCount: integer read FEpochCount write SetEpochCount;
    property Momentum: double read FMomentum write SetMomentum;
    property NeuronsInLayer: TStrings read FNeuronsInLayer write
FNeuronsInLayer;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnAfterNeuronCreated: TNotifyEvent read FOnAfterNeuronCreated write
FOnAfterNeuronCreated;
    property OnAfterTeach: TNotifyEvent read FOnAfterTeach write FOnAfterTeach;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnBeforeTeach: TNotifyEvent read FOnBeforeTeach write
FOnBeforeTeach;
```

```
    property OnEpochPassed: TNotifyEvent read FOnEpochPassed write
FOnEpochPassed;
    property PatternCount: integer read FPatternCount write SetPatternCount;
    property TeachRate: double read FTeachRate write SetTeachRate;
  end;

  // Клас мереж системи кібербезпеки для моделювання ботнет мережіі back-
propagation TNeuralNetExtended }
  TNeuralNetExtended = class(TNeuralNetBP)
  private
    // Файл даних
    FNeuroDataSource: TNeuroDataSource;
    // Ім'я файлу даних *.txt
    FSourceFileName: TFileName;
    // Ім'я конфігураційного файлу *.nnw
    FFileName: TFileName;
    // Конфігураційний файл
    FNnwFile: TIniFile;
    // Поля
    FFields: TNeuroFields;
    // Кількість доступних полів
    FAvailableFieldsCount: integer;
    FMaxTeachError: boolean;
    FMaxTeachErrorValue: double;
    FMaxTestError: boolean;
    FMaxTestErrorValue: double;
    FMidTeachError: boolean;
    FMidTeachErrorValue: double;
    FMidTestError: boolean;
    FMidTestErrorValue: double;
    FOptions: string;
    FSettingsLoaded: boolean;
    FTestAsValid: boolean;
    FTeachIdent: boolean;
    FTeachIdentCount: integer;
    FTestIdent: boolean;
    FTestIdentCount: integer;
    FUseForTeach: integer;
    FIdentError: double;
    FRealOutputIndex: TVectorInt;
    FRealInputIndex: TVectorInt;
    function GetFields(Index: integer): TNeuroField;
    function GetInputFieldCount: integer;
    function GetOutputFieldCount: integer;
    function GetRealInputIndex(Index: integer): integer;
    function GetRealOutputIndex(Index: integer): integer;
    procedure SetFields(Index: integer; Value: TNeuroField);
    procedure SetFileName(Value: TFilename);
    procedure SetAvailableFieldsCount(Value: integer);
    procedure SetUseForTeach(const Value: integer);
    procedure SetTeachIdentCount(const Value: integer);
    procedure SetRealOutputIndex(Index: integer; const Value: integer);
    procedure SetRealOutputIndexCount(const Value: integer);
    procedure SetRealInputIndex(Index: integer; const Value: integer);
    procedure SetRealInputIndexCount(const Value: integer);
  protected
    function GetOutput(Index: integer): double; override;
    procedure DoOnBeforeTeach; override;
    procedure DoOnEpochPassed; override;
    procedure SetDefaultProperties; override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure ComputeUnPrepData(AVector: TVectorFloat);
    // Завантажує дані з текстового файлу
    procedure LoadDataFrom;
    // Завантажує настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure LoadNetwork;
```

```
    // Завантажує настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure LoadPhase1;
    // Завантажує настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure LoadPhase2;
    // Завантажує настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure LoadPhase4;
    // Нормалізує набір даних
    procedure NormalizeData;
    // Зберігає настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure SaveNetwork;
    // Зберігає настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure SavePhase1;
    // Зберігає настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure SavePhase2;
    // Зберігає настроювання мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure SavePhase4;
    // Навчання нейронної мереж системи кібербезпеки для моделювання ботнет
мережіі
    procedure Train;
    property AvailableFieldsCount: integer read FAvailableFieldsCount write
SetAvailableFieldsCount;
    property Fields[Index: integer]: TNeuroField read GetFields write SetFields;
    property InputFieldCount: integer read GetInputFieldCount;
    property OutputFieldCount: integer read GetOutputFieldCount;
    property SettingsLoaded: boolean read FSettingsLoaded write FSettingsLoaded;
    property RealOutputIndex[Index: integer]: integer read GetRealOutputIndex
write SetRealOutputIndex;
    property RealOutputIndexCount: integer write SetRealOutputIndexCount;
    property RealInputIndex[Index: integer]: integer read GetRealInputIndex
write SetRealInputIndex;
    property RealInputIndexCount: integer write SetRealInputIndexCount;
    property NnwFile: TIniFile read FNnwFile write FNnwFile;
  published
    property FileName: TFileName read FFileName write SetFileName;
    property IdentError: double read FIdentError write FIdentError;
    property MaxTeachError: boolean read FMaxTeachError write FMaxTeachError;
    property MaxTeachErrorValue: double read FMaxTeachErrorValue write
FMaxTeachErrorValue;
    property MaxTestError: boolean read FMaxTestError write FMaxTestError;
    property MaxTestErrorValue: double read FMaxTestErrorValue write
FMaxTestErrorValue;
    property MidTeachError: boolean read FMidTeachError write FMidTeachError;
    property MidTeachErrorValue: double read FMidTeachErrorValue write
FMidTeachErrorValue;
    property MidTestError: boolean read FMidTestError write FMidTestError;
    property MidTestErrorValue: double read FMidTestErrorValue write
FMidTestErrorValue;
    property Options: string read FOptions write FOptions;
    property SourceFileName: TFileName read FSourceFileName write
FSourceFileName;
    property TestAsValid: boolean read FTestAsValid write FTestAsValid;
    property TeachIdent: boolean read FTeachIdent write FTeachIdent;
    property TeachIdentCount: integer read FTeachIdentCount write
SetTeachIdentCount;
    property TestIdent: boolean read FTestIdent write FTestIdent;
    property TestIdentCount: integer read FTestIdentCount write FTestIdentCount;
    property UseForTeach: integer read FUseForTeach write SetUseForTeach;
  end;


procedure Register;
```

```
implementation

{$R *.RES}

{ TNeuron }

constructor TNeuron.Create(ALayer: TLayer);
begin
  inherited Create;
  // покажчик на шар у якому перебуває нейрон
  Layer := ALayer;
end;

destructor TNeuron.Destroy;
begin
  WeightCount := 0;
  FWeights := nil;
  Layer := nil;
  inherited;
end;

procedure TNeuron.ComputeOut(const AInputs: TVectorFloat);
var
  i: integer;
begin
  FOutput := 0;
  // Підраховується зважена сума нейрона
  for i := Low(AInputs) to High(AInputs) do
    FOutput := FOutput + FWeights[i] * AInputs[i];
end;

function TNeuron.GetWeights(Index: integer): double;
begin
  try
    Result := FWeights[Index];
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

procedure TNeuron.InitWeights;
var
  i: integer;
begin
  // Ініціалізація ваг нейрона
  for i := Low(FWeights) to High(FWeights) do
    FWeights[i] := Random
end;

procedure TNeuron.SetWeightCount(const Value: integer);
begin
  SetLength(FWeights, Value);
end;

procedure TNeuron.SetWeights(Index: integer; Value: double);
begin
  try
    FWeights[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

{ Кінець опису TNeuron }

{ TNeuronHopf }
```

```
procedure TNeuronHopf.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // гранична функція
  if FOutput >= 0 then
    FOutput := 1
  else
    FOutput := -1
end;

{ Кінець опису TNeuronHopf}

{ TNeuronBP }

destructor TNeuronBP.Destroy;
begin
  FOnActivation := nil;
  FOnActivation := nil;
  PrevUpdateCount := 0;
  FPrevUpdate := nil;
  inherited;
end;

function TNeuronBP.GetLearningRate(Index: integer): double;
begin
  Result := FLearningRate[Index];
end;

function TNeuronBP.GetPrevDerivative(Index: integer): double;
begin
  Result := FPrevDerivative[Index];
end;

function TNeuronBP.GetPrevUpdateCount: integer;
begin
  Result := High(FPrevUpdate) + 1;
end;

function TNeuronBP.GetPrevUpdate(Index: integer): double;
begin
  Result := FPrevUpdate[Index];
end;

procedure TNeuronBP.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // Задає зсув нейрона
  FOutput := FOutput + Weights[High(AInputs) + 1];
  FOutput := OnActivation(FOutput);
end;

procedure TNeuronBP.SetDelta(Value: double);
begin
  FDelta := Value;
end;

procedure TNeuronBP.SetLearningRate(Index: integer; const Value: double);
begin
  FLearningRate[Index] := Value;
end;

procedure TNeuronBP.SetLearningRateCount(const Value: integer);
begin
  SetLength(FLearningRate, Value)
end;

procedure TNeuronBP.SetPrevUpdate(Index: integer; Value: double);
begin
  FPrevUpdate[Index] := Value;
```

```
end;

procedure TNeuronBP.SetPrevUpdateCount(const Value: integer);
begin
  SetLength(FPrevUpdate, Value)
end;

procedure TNeuronBP.SetPrevDerivative(Index: integer; const Value: double);
begin
  FPrevDerivative[Index] := Value;
end;

procedure TNeuronBP.SetPrevDerivativeCount(const Value: integer);
begin
  SetLength(FPrevDerivative, Value)
end;

{ Кінець опису TNeuronBP }

{ TLayer }

procedure TLayer.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayer).FNumber;
  NeuronCount := (Source as TLayer).NeuronCount;
  // Створюються нейрони
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

constructor TLayer.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  inherited Create;
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

destructor TLayer.Destroy;
var
  i: integer;
begin
  for i := 0 to NeuronCount - 1 do
    FNeurons[i].Free;
  NeuronCount := 0;
  FNeurons := nil;
  inherited;
end;

function TLayer.GetNeuronCount: integer;
begin
  Result := High(FNeurons) + 1;
end;

function TLayer.GetNeurons(Index: integer): TNeuron;
begin
  Result := FNeurons[Index];
end;

procedure TLayer.SetNeuronCount(Value: integer);
begin
  if Value <> High(FNeurons) + 1 then
    SetLength(FNeurons, Value);
end;
```

```
procedure TLayer.SetNeurons(Index: integer; Value: TNeuron);
begin
  try
    FNeurons[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

{ TLayerHopf }

constructor TLayerHopf.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronHopf.Create(Self);
end;

{ TLayerBP }

procedure TLayerBP.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayerBP).FNumber;
  NeuronCount := (Source as TLayerBP).NeuronCount;
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

constructor TLayerBP.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

destructor TLayerBP.Destroy;
begin
  inherited;
end;

function TLayerBP.GetNeuronsBP(Index: integer): TNeuronBP;
begin
  Result := FNeurons[Index] as TNeuronBP;
end;

procedure TLayerBP.SetNeuronsBP(Index: integer; Value: TNeuronBP);
begin
  FNeurons[Index] := Value as TNeuronBP;
end;

{ TNeuralNet }

destructor TNeuralNet.Destroy;
begin
  Clear;
  SetLength(FPatternsInput, 0, 0);
  FPatternsInput := nil;
  SetLength(FPatternsOutput, 0 ,0);
  FPatternsOutput := nil;
```

```
  FLayers := nil;
  inherited;
end;

procedure TNeuralNet.Clear;
var
  i, xCount: integer;
begin
  xCount := LayerCount;
  if  xCount > 0 then
  begin
    for i := 0 to xCount - 1 do
      FLayers[i].Free;
    LayerCount := 0;
  end;
end;

function TNeuralNet.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNet.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNet.GetLayers(Index: integer): TLayer;
begin
  Result := FLayers[Index];
end;

function TNeuralNet.GetOutputNeuronCount: integer;
begin
  Result := Layers[LayerCount - 1].NeuronCount;
end;

function TNeuralNet.GetPatternsInput(PatternIndex: integer; InputIndex:
integer): double;
begin
  Result := FPatternsInput[PatternIndex, InputIndex];
end;

procedure TNeuralNet.AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AInputs) + 1 then
    raise EInOutDimensionError.Create(SInVectorCount);
  if OutputNeuronCount <> High(AOutputs) + 1 then
    raise EInOutDimensionError.Create(SOutVectorCount);
  PatternCount := PatternCount + 1;
  ResizeInputDim;
  ResizeOutputDim;
  for i := Low(AInputs) to High(AInputs) do
    PatternsInput[PatternCount - 1, i] := AInputs[i];
  for i := Low(AOutputs) to High(AOutputs) do
    PatternsOutput[PatternCount - 1, i] := AOutputs[i];
end;

procedure TNeuralNet.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    // видаляє вхідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to InputNeuronCount - 1 do
```

```
        FPatternsInput[i, j] := FPatternsInput[i + 1, j];
    // видаляє вихідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to OutputNeuronCount - 1 do
        FPatternsOutput[i, j] := FPatternsOutput[i + 1, j];
    Dec(FPatternCount);
    ResizeInputDim;
    ResizeOutputDim;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNet.ResetPatterns;
begin
  FPatternCount := DefaultPatternCount;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternsOutput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsOutput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.SetPatternsInput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsInput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.Init(const ANeuronsInLayer: TVectorInt);
var
  i,j: integer;
begin
  LayerCount := High(ANeuronsInLayer) + 1;
  // FLayers[0] нульовий шар і виконує роль розподільного,
  // використовується тільки поле Output
  FLayers[0] := TLayer.Create(0, ANeuronsInLayer[0]);
  // для нульового шару не потрібні вагові коефіцієнти
  for i := 1 to LayerCount - 1 do
  begin
    FLayers[i] := TLayer.Create(i, ANeuronsInLayer[i]);
    for j := 0 to ANeuronsInLayer[i] - 1 do
    with FLayers[i].FNeurons[j] do
      // задає кількість елементів у векторі ваг нейрона j в
      //  шарі і рівним кількості виходів попереднього шару
      WeightCount := FLayers[ i-1].NeuronCount;
  end;
end;

procedure TNeuralNet.ResizeInputDim;
begin
  SetLength(FPatternsInput, FPatternCount, InputNeuronCount)
end;

procedure TNeuralNet.ResizeOutputDim;
```

```
begin
  SetLength(FPatternsOutput, FPatternCount, OutputNeuronCount)
end;

procedure TNeuralNet.SetLayerCount(Value: integer);
begin
  SetLength(FLayers, Value);
end;

procedure TNeuralNet.SetLayers(Index: integer; Value: TLayer);
begin
  try
    FLayers[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

{  TNeuralNetHopf }

constructor TNeuralNetHopf.Create(AOwner: TComponent);
begin
  inherited;
  PatternCount := DefaultPatternCount;
  InputNeuronCount := DefaultNeuronCount;
  MaxIterCount := DefaultMaxIterCount;
  AutoInit := False;
end;

destructor TNeuralNetHopf.Destroy;
begin
  FOnAfterInit := nil;
  FOnBeforeInit := nil;
  FOnPatternRecognized := nil;
  SetLength(FPatterns, 0, 0);
  FPatterns := nil;
  inherited;
end;

function TNeuralNetHopf.GetInput(Index: integer): double;
begin
  Result := Layers[1].Neurons[Index].Output;
end;

function TNeuralNetHopf.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNetHopf.GetLayerCount: integer;
begin
  Result := DefaultHopfLayerCount;
end;

function TNeuralNetHopf.GetPatterns(InputIndex: integer; PatternIndex: integer):
integer;
begin
  Result := FPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetHopf.Stabled: boolean;
var
  i: integer;
begin
  // Порівнює вихідні значення попередньої
  //  ітерації зі значеннями поточної
  Result := True;
  for i := 0 to InputNeuronCount - 1  do
```

```
    if FLayers[1].FNeurons[i].FOutput <> FLayers[0].FNeurons[i].FOutput then
    begin
      Result := False;
      Exit
    end;
end;

procedure TNeuralNetHopf.AddPattern(const ANewPattern: TVectorInt);
var
  i: integer;
begin
  if InputNeuronCount <> High(ANewPattern)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  PatternCount := PatternCount + 1;
  ResizePatternsDim;
  for i := 0 to FInputNeuronCount - 1 do
    FPatterns[FPatternCount - 1, i] := ANewPattern[i];
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.Calc;
var
  i: integer;
  xCurrentIter: integer;
  xArray: TVectorFloat;
begin
  SetLength(xArray, InputNeuronCount);
  // Цикл працює поки не стабілізуються виходи
  xCurrentIter := 0;
  repeat
    for i := 0 to InputNeuronCount - 1 do
    begin
      // Запам'ятовує попередній крок ітерації, для
      //  цього використовується нульовий шар
      Layers[SensorLayer].Neurons[i].Output := Layers[1].Neurons[i].Output;
      xArray[i] := Layers[1].Neurons[i].Output;
    end;
    for i := 0 to InputNeuronCount - 1 do
      with Layers[1].Neurons[i] do
        // Розраховується новий стан нейронів і аксонів
        ComputeOut(xArray);
    Inc(xCurrentIter);
  until Stabled or (MaxIterCount = xCurrentIter);
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetHopf.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    for i := Index to FPatternCount - 2 do
      for j := 0 to FInputNeuronCount - 1 do
        FPatterns[i, j] := FPatterns[i + 1, j];
    Dec(FPatternCount);
    ResizePatternsDim;
    if AutoInit then
      InitWeights;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.Init;
```

```
var
  i, j: integer;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
  LayerCount := DefaultHopfLayerCount;
  for i := 0 to LayerCount - 1 do
    FLayers[i] := TLayerHopf.Create(i, FInputNeuronCount);
  // Для нульового шару не потрібні вагові коефіцієнти
  for j := 0 to FInputNeuronCount - 1 do
    with FLayers[1].FNeurons[j] do
      // задає кількість елементів у векторі
      WeightCount := FInputNeuronCount;
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetHopf.InitWeights;
var
  i, j, k : integer;
begin
  // Ініціалізує вагову матрицю
  for i := 0 to InputNeuronCount - 1 do
    for j := 0 to InputNeuronCount - 1 do
    with Layers[1].Neurons[i] do
    begin
      Weights[j] := 0;
      if i <> j then
        for k := 0 to PatternCount - 1 do
          Weights[j] := Weights[j]+ Patterns[k, i] * Patterns[k, j]
    end;
end;

procedure TNeuralNetHopf.ResetPatterns;
begin
  PatternCount := DefaultPatternCount;
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.ResizePatternsDim;
begin
  SetLength(FPatterns, FPatternCount, FInputNeuronCount);
end;

procedure TNeuralNetHopf.SetInput(Index: integer; Value: double);
begin
  try
    Layers[1].Neurons[Index].Output := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.SetInputNeuronCount(Value: integer);
begin
  if Value > DefaultNeuronCount then
    FInputNeuronCount := Value
  else
    FInputNeuronCount := DefaultNeuronCount;
  ResizePatternsDim;
  Init;
end;

procedure TNeuralNetHopf.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
```

```
  else
    FPatternCount := Value;
end;

procedure TNeuralNetHopf.SetPatterns(InputIndex: integer; PatternIndex: integer;
Value: integer);
begin
  FPatterns[InputIndex, PatternIndex] := Value;
end;

{ TNeuralNetBP }

constructor TNeuralNetBP.Create(AOwner: TComponent);
var
  i: integer;
begin
  inherited;
  FNeuronsInLayer := TStringList.Create;
  for i := 0 to DefaultLayerCount do
    AddLayer(DefaultNeuronCount);
  TStringList(FNeuronsInLayer).OnChange := NeuronsInLayerChange;
  AutoInit := True;
  StopTeach := False;
  TeachStopped := False;
  NeuronsInLayerChange(Self);
  SetDefaultProperties;
end;

destructor TNeuralNetBP.Destroy;
begin
  FNeuronsInLayer.Free;
  SetLength(FRandomOrder, 0);
  FRandomOrder := nil;
  SetLength(FDesiredOut, 0);
  FDesiredOut := nil;
  SetLength(FTestSetPatterns, 0, 0);
  FTestSetPatterns := nil;
  SetLength(FTestSetPatternsOut, 0, 0);
  FTestSetPatternsOut := nil;
  FOnAfterInit := nil;
  FOnAfterTeach := nil;
  FOnBeforeInit := nil;
  FOnBeforeTeach := nil;
  FOnEpochPassed := nil;
  inherited;
end;

function TNeuralNetBP.GetLayersBP(Index: integer): TLayerBP;
begin
  Result := FLayers[Index] as TLayerBP;
end;

function TNeuralNetBP.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNetBP.GetDesiredOut(Index: integer): double;
begin
  Result := FDesiredOut[Index];
end;

function TNeuralNetBP.GetOutput(Index: integer): double;
begin
  try
    Result := LayersBP[LayerCount - 1].NeuronsBP[Index].Output;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
```

```
  end;
end;

function TNeuralNet.GetPatternsOutput(PatternIndex: integer; OutputIndex:
integer): double;
begin
  Result := FPatternsOutput[PatternIndex, OutputIndex];
end;

function TNeuralNetBP.QuadError: double;
var
  i: integer;
begin
  // розраховує середньоквадратичну помилку
  Result := 0;
  for i := 0 to OutputNeuronCount - 1 do
    Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
DesiredOut[i]);
  Result := Result/2;
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Активаційна функція - сигмоїд
  Result := 1/( 1 + exp(-FAlpha * Value) )
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Похідна сигмоїди
  Result := FAlpha * Value * (1 - Value)
end;

function TNeuralNetBP.GetTestSetPatterns(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetBP.GetTestSetPatternsOut(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatternsOut[InputIndex, PatternIndex];
end;

procedure TNeuralNetBP.AddLayer(ANeurons: integer);
begin
  if ANeurons < DefaultNeuronCount then
    NeuronCountError
  else
    NeuronsInLayer.Add(IntToStr(ANeurons));
end;

procedure TNeuralNetBP.AdjustWeights;
var
  i, j, k: integer;
  xCurrentUpdate: double;
begin
  // Підстроювання ваг починаючи з першого шару
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount  do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // коректує вагу з'єднуючого j-нейрона шару i
        //  з k-нейроном шару i-1:  добутком дельта j-нейрона
        //  на вихід k-нейрона шару i-1
        if k = LayersBP[ i-1].NeuronCount then
```

```
            // якщо це нейрон, що задає зсув
            xCurrentUpdate := -TeachRate * Delta + Momentum * PrevUpdate[k]
          else
            xCurrentUpdate := -TeachRate * Delta *
            LayersBP[ i-1].NeuronsBP[k].Output + Momentum * PrevUpdate[k];
          Weights[k]:= Weights[k] + xCurrentUpdate;
          PrevUpdate[k] := xCurrentUpdate;
        end;
      end
end;

procedure TNeuralNetBP.CalcLocalError;
var
  i, j, k: integer;
begin
  // Дельта-правило з останнього шару до першого
  for i := LayerCount - 1 downto 1 do
    // для останнього шару
    if i = LayerCount - 1 then
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Delta := (LayersBP[i].NeuronsBP[j].Output-
DesiredOut[j])
        * Activation(LayersBP[i].NeuronsBP[j].Output)
    else
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        with LayersBP[i].NeuronsBP[j] do
        begin
          Delta := 0;
          // Підсумує добуток локальної помилки k-нейрона шару i+1
          //  на вагу з'єднуючий k-нейрон шару i+1 з j-нейроном шару i
          for k := 0 to LayersBP[i+1].NeuronCount - 1 do
            Delta := Delta + LayersBP[i+1].NeuronsBP[k].Delta *
                             LayersBP[i+1].NeuronsBP[k].Weights[j];
          Delta := Delta * Activation(Output)
        end;
end;

procedure TNeuralNetBP.CheckTestSet;
var
  i, j: integer;
  xArray: TVectorFloat;
  xFirstTestSample: boolean;
  xQuadError: double;
  // функція розраховує середньоквадратичну помилку
  function QuadError(APatternCount: integer): double;
  var
    i: integer;
  begin
    Result := 0;
    for i := 0 to OutputNeuronCount - 1 do
      Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
TestSetPatternsOut[APatternCount, i]);
    Result := Result/2;
  end;
begin
  SetLength(xArray, InputNeuronCount);
  xFirstTestSample := True;
  FRecognizedTestCount := 0;
  FMidTestResidual := 0;
  FMaxTestResidual := 0;
  for i := 0 to TestSetPatternCount - 1 do
  begin
    for j := 0 to InputNeuronCount - 1 do
      xArray[j] := TestSetPatterns[i, j];
    Compute(xArray);
    xQuadError := QuadError(i);
    // перевірка - чи розпізнаний приклад з тестової множини
    if xQuadError < IdentError then
      Inc(FRecognizedTestCount);
```

```
    FMidTestResidual := FMidTestResidual + xQuadError;
    // максимальна помилка на тестовій множині
    if xFirstTestSample then
    begin
      FMaxTestResidual := xQuadError;
      xFirstTestSample := False;
    end
    else
      if FMaxTestResidual < xQuadError then
        FMaxTestResidual := xQuadError;
  end;
  // середня помилка на тестовій множині
  FMidTestResidual := FMidTestResidual/TestSetPatternCount;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.Compute(AVector: TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  AVector[i];
  Propagate;
end;

procedure TNeuralNetBP.DoOnAfterInit;
begin
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetBP.DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex:
integer);
var
  i: integer;
begin
  with LayersBP[ALayerIndex].NeuronsBP[ANeuronIndex] do
    for i := 0 to PrevUpdateCount - 1 do
      PrevUpdate[i] := 0;
  if Assigned(FOnAfterNeuronCreated) then
    FOnAfterNeuronCreated(Self);
end;

procedure TNeuralNetBP.DoOnAfterTeach;
begin
  if Assigned(FOnAfterTeach) then
    FOnAfterTeach(Self);
end;

procedure TNeuralNetBP.DoOnBeforeInit;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
end;

procedure TNeuralNetBP.DoOnBeforeTeach;
begin
  if Assigned(FOnBeforeTeach) then
    FOnBeforeTeach(Self);
end;

procedure TNeuralNetBP.DoOnEpochPassed;
begin
  if Assigned(FOnEpochPassed) then
    FOnEpochPassed(Self);
end;
```

```
procedure TNeuralNetBP.DeleteLayer(Index: integer);
var
  i: integer;
begin
  try
    NeuronsInLayer.Delete(Index);
    for i := Index to LayerCount - 2 do
      LayersBP[i].Assign(LayersBP[i + 1]);
    FLayers[LayerCount - 1].Free;
    LayerCount := LayerCount - 1;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

procedure TNeuralNetBP.Init;
var
  i, j: integer;
begin
  DoOnBeforeInit;
  if NeuronsInLayer.Count > 0 then
  begin
  LayerCount := NeuronsInLayer.Count;
    // FLayers[0] нульовий шар, використовується тільки поле Output
    FLayers[0] := TLayerBP.Create(0, StrToInt(NeuronsInLayer.Strings[0]));
    // для нульового шару не потрібні вагові коефіцієнти
    for i := 1 to LayerCount - 1 do
    begin
      FLayers[i] := TLayerBP.Create(i, StrToInt(NeuronsInLayer.Strings[i]));
      for j := 0 to StrToInt(NeuronsInLayer.Strings[i]) - 1 do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // задає кількість елементів у векторі ваг + зсув
        WeightCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        // задає кількість  у векторі утримуючих попередню
        //  корекцію елементів  + зсув
        PrevUpdateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        PrevDerivativeCount := LayersBP[ i-1].NeuronCount + BiasNeuron; // для
швидких алгоритмів
        LearningRateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;   // для
швидких алгоритмів
        OnActivation := Activation;
        OnActivation := Activation;
        Randomize;
        DoOnAfterNeuronCreated(i, j);
      end
    end;
    // установлює розмірність масиву виходів
    //  число нейронів в останньому шарі = числу виходів
    SetLength(FDesiredOut, OutputNeuronCount);
  end;
  DoOnAfterInit;
end;

procedure TNeuralNetBP.InitWeights;
var
  i,j: integer;
begin
  Randomize;
  // Ініціалізація ваг
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
       LayersBP[i].NeuronsBP[j].InitWeights;
end;

procedure TNeuralNetBP.LoadPatternsInput(APatternIndex :integer);
var
```

```
  i: integer;
begin
  for i := 0 to InputNeuronCount - 1 do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  PatternsInput[APatternIndex,
i];
end;

procedure TNeuralNetBP.LoadPatternsOutput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to OutputNeuronCount - 1 do
    DesiredOut[i] := PatternsOutput[APatternIndex, i];
end;

procedure TNeuralNetBP.NeuronsInLayerChange(Sender: TObject);
begin
  if AutoInit then
    Init;
end;

procedure TNeuralNetBP.NeuronCountError;
begin
  raise ENeuronCountError.Create(SNeuronCount)
end;

procedure TNeuralNetBP.Propagate;
var
  i, j, xIndex: integer;
  xArray: TVectorFloat;
begin
   // Поширення сигналу в прямому напрямку з першого шару
  for i := 1 to LayerCount - 1 do
  begin
    // формування масиву входів з виходів попереднього шару
    SetLength(xArray, LayersBP[ i-1].NeuronCount);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := LayersBP[ i-1].NeuronsBP[xIndex].Output;
    // обчислення виходу нейрона
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      with LayersBP[i].NeuronsBP[j] do
        ComputeOut(xArray);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := 0;
  end;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.ResetLayers;
begin
  Clear;
  FNeuronsInLayer.Clear;
end;

procedure TNeuralNetBP.SetDesiredOut(Index: integer; Value: double);
begin
  FDesiredOut[Index] := Value;
end;

procedure TNeuralNetBP.SetLayersBP(Index: integer; Value: TLayerBP);
begin
  FLayers[Index] := Value as TLayerBP;
end;

procedure TNeuralNetBP.SetAlpha(Value: double);
begin
  if (Value > 10) or (Value < 0.01) then
    FAlpha := DefaultAlpha
```

```
  else
    FAlpha := Value;
end;

procedure TNeuralNetBP.SetTeachRate(Value: double);
begin
  if (Value > 1) or (Value <= 0) then
    FTeachRate := DefaultTeachRate
  else
    FTeachRate := Value;
end;

procedure TNeuralNetBP.SetTestSetPatterns(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatterns[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternsOut(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatternsOut[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternCount(const Value: integer);
begin
  FTestSetPatternCount := Value;
  SetLength(FTestSetPatterns, FTestSetPatternCount, InputNeuronCount);
  SetLength(FTestSetPatternsOut, FTestSetPatternCount, OutputNeuronCount);
end;

procedure TNeuralNetBP.SetMomentum(Value: double);
begin
  if (Value > 1) or (Value < 0) then
    FMomentum := DefaultMomentum
  else
    FMomentum := Value;
end;

procedure TNeuralNetBP.SetEpochCount(Value: integer);
begin
  if Value < 1 then
    FEpochCount := 1
  else
    FEpochCount := Value;
end;

procedure TNeuralNetBP.ShakeUp;
var
  i, j, k: integer;
begin
  Randomize;
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      for k := 0 to LayersBP[ i-1].NeuronCount  do
        with LayersBP[i].NeuronsBP[j] do
          Weights[k]:= Weights[k] + Random*0.1-0.05;
end;

procedure TNeuralNetBP.Shuffle;
var
  i, j, xNewInd, xLast: integer;
  xIsUnique : boolean;
begin
  xNewInd := 0;
  FRandomOrder[0] := Round(Random(FPatternCount));
  xLast := 0;
  for i := 1 to PatternCount - 1 do
  begin
```

```
    xIsUnique := False;
    while not xIsUnique do
    begin
      xNewInd := Round((Random(FPatternCount)));
      xIsUnique := True;
      for j := 0 to xLast do
        if xNewInd = FRandomOrder[j] then
          xIsUnique := False;
    end;
    FRandomOrder[i] := xNewInd;
    xLast := xLast +1;
  end;
end;

procedure TNeuralNetBP.TeachOffLine;
var
  j: integer;
  xQuadError: double;
  xNewEpoch: boolean;
begin
  DoOnBeforeTeach;
  if not ContinueTeach then
  begin
    // ваги ініціалізуються, якщо мереж системи кібербезпеки для моделювання
ботнет мережіа навчається з "нуля"
    InitWeights;
    FEpochCurrent := 1;
  end;
  Randomize;
  SetLength(FRandomOrder, FPatternCount);
  TeachStopped := False;
  while (FEpochCurrent <= EpochCount) do
  begin
    FTeachError := 0;
    FMaxTeachResidual := 0;
    FRecognizedTeachCount := 0;
    xNewEpoch := True;
    Shuffle;
    for j := 0 to PatternCount - 1 do
    begin
      LoadPatternsInput(FRandomOrder[j]);
      LoadPatternsOutput(FRandomOrder[j]);
      Propagate;
      xQuadError := QuadError;
      // перевірка - чи розпізнаний приклад з навчальної множини
      if xQuadError < IdentError then
        Inc(FRecognizedTeachCount);
      FTeachError := FTeachError + xQuadError;
      // максимальна помилка на навчальній множині
      if xNewEpoch then
      begin
        FMaxTeachResidual := xQuadError;
        xNewEpoch := False;
      end
      else
        if MaxTeachResidual < xQuadError then
          FMaxTeachResidual := xQuadError;
      CalcLocalError;
      AdjustWeights;
    end;
    // середня помилка на навчальній множині
    FMidTeachResidual := TeachError/PatternCount;
    // перевірка мереж системи кібербезпеки для моделювання ботнет мережіі на
узагальнення
    if TestSetPatternCount > 0 then
      CheckTestSet;
    DoOnEpochPassed;
    if StopTeach then
    begin
```

```
      TeachStopped := True;
      Exit;
    end;
    Inc(FEpochCurrent);
  end;
  DoOnAfterTeach;
end;

procedure TNeuralNetBP.SetPatternCount(const Value: integer);
begin
  FPatternCount := Value;
  inherited;
end;

procedure TNeuralNetBP.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Alpha := DefaultAlpha;
  ContinueTeach := False;
  Epoch := True;
  EpochCount := DefaultEpochCount;
  Momentum := DefaultMomentum;
  TeachRate := DefaultTeachRate;
  ResizeInputDim;
  ResizeOutputDim;
end;

{ TNeuralNetExtended }

constructor TNeuralNetExtended.Create(AOwner: TComponent);
begin
  inherited;
  SetDefaultProperties;
end;

destructor TNeuralNetExtended.Destroy;
var
  i: integer;
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  FNeuroDataSource.Free;
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i].Free;
  inherited;
end;

function TNeuralNetExtended.GetFields(Index: integer): TNeuroField;
begin
  Result := FFields[Index];
end;

function TNeuralNetExtended.GetInputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdInput then
      Inc(Result);
end;

function TNeuralNetExtended.GetOutputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdOutput then
```

```
      Inc(Result);
end;

function TNeuralNetExtended.GetOutput(Index: integer): double;
var
  xTmp: double;
begin
  with Fields[RealOutputIndex[Index]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := -ln(1/LayersBP[LayerCount - 1].NeuronsBP[Index].Output -
1);
                 LayersBP[LayerCount - 1].NeuronsBP[Index].Output := xTmp *
Dispersion + ValueMid;
               end;
      nrmLinear:  Result := (LayersBP[LayerCount - 1].NeuronsBP[Index].Output +
1)*(ValueMax - ValueMin)/2 + ValueMin;
      nrmLinearOut: Result := LayersBP[LayerCount -
1].NeuronsBP[Index].Output*(ValueMax - ValueMin) + ValueMin;
      nrmSigmoid:  Result := - Ln(1/LayersBP[LayerCount -
1].NeuronsBP[Index].Output - 1)/Alpha;
    end;
end;

function TNeuralNetExtended.GetRealInputIndex(Index: integer): integer;
begin
  Result := FRealInputIndex[Index];
end;

function TNeuralNetExtended.GetRealOutputIndex(Index: integer): integer;
begin
  Result := FRealOutputIndex[Index];
end;

procedure TNeuralNetExtended.ComputeUnPrepData(AVector: TVectorFloat);
var
  i: integer;
  xTmp: double;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
  with FFields[RealInputIndex[i]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := (LayersBP[SensorLayer].NeuronsBP[i].Output -
ValueMid)/Dispersion;
                 LayersBP[SensorLayer].NeuronsBP[i].Output := 1/(1 + exp(-
xTmp));
               end;
      nrmLinear: LayersBP[SensorLayer].NeuronsBP[i].Output := 2*(AVector[i] -
ValueMin)/(ValueMax - ValueMin) - 1;
      nrmLinearOut: LayersBP[SensorLayer].NeuronsBP[i].Output := (AVector[i] -
ValueMin)/(ValueMax - ValueMin);
      nrmSigmoid: LayersBP[SensorLayer].NeuronsBP[i].Output := 1/( 1 + exp(-
Alpha * AVector[i]));
    end;
  Propagate;
end;

procedure TNeuralNetExtended.DoOnBeforeTeach;
begin
  if InputNeuronCount <> InputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SInFieldCount);
  if OutputNeuronCount <> OutputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SOutFieldCount);
  if InputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  if OutputNeuronCount < 0 then
```

```
    raise EInOutDimensionError.Create(SOutNeuronCount);
  if (not FMaxTeachError) and (not FMaxTestError) and
     (not FMidTeachError) and (not FMidTestError) and (not FEpoch) then
    raise EBPStopCondition.Create(SBPStopCondition);
  inherited DoOnBeforeTeach;
end;

procedure TNeuralNetExtended.DoOnEpochPassed;
begin
  if MaxTeachError and (MaxTeachResidual < MaxTeachErrorValue) then
    StopTeach := True;
  if MidTeachError and (MidTeachResidual < MidTeachErrorValue) then
    StopTeach := True;
  if MaxTestError and (MaxTestResidual < MaxTestErrorValue) then
    StopTeach := True;
  if MidTestError and (MidTestResidual < MidTestErrorValue) then
    StopTeach := True;
  if TeachIdent and (Round((FRecognizedTeachCount * 100)/PatternCount) <
TeachIdentCount) then
    StopTeach := True;
  if TestIdent and (Round((FRecognizedTestCount * 100)/TestSetPatternCount) <
TestIdentCount) then
    StopTeach := True;
  inherited DoOnEpochPassed;
end;

procedure TNeuralNetExtended.LoadDataFrom;
var
  xTempStream: TFileStream;
  i, j: integer;
  xFieldCount: integer;
  xArray: TVectorFloat;
  xPatternsList: TStringList;
begin
  // створюється потік
  xTempStream := TFileStream.Create(FSourceFileName, fmOpenRead);
  // створюється список
  xPatternsList := TStringList.Create;
  xPatternsList.LoadFromStream(xTempStream);
  try
    if SettingsLoaded then
    begin
      xFieldCount := FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
      if AvailableFieldsCount <> xFieldCount then
        if MessageDlg('Кількість полів у файлі даних не відповідає значенню
AvailableFieldsCount'+ #13 + 'Установити нове значення AvailableFieldsCount = '+
IntToStr(xFieldCount),
                      mtConfirmation, [mbYes, mbNo], 0) = mrYes then
          AvailableFieldsCount := xFieldCount;
    end
    else
      AvailableFieldsCount :=
FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
    FNeuroDataSource.ExtractHeaders(FFields, xPatternsList.Strings[0]);
    // установлюється розмірність часового масиву
    SetLength(xArray, FAvailableFieldsCount);
    // установлюється розмірність масиву даних
    if FUseForTeach = 100 then
      PatternCount := xPatternsList.Count - 1
    else
    begin
      PatternCount := Round((xPatternsList.Count - 1) * FUseForTeach / 100);
      TestSetPatternCount := xPatternsList.Count - PatternCount - 1;
    end;
    for i := 0 to FAvailableFieldsCount - 1 do
      FFields[i].DataInCount := xPatternsList.Count - 1;
    for j := 0 to xPatternsList.Count - 2 do
    begin
      FNeuroDataSource.ExtractValues(xArray, xPatternsList.Strings[j + 1]);
```

```
      for i := 0 to FAvailableFieldsCount - 1 do
        FFields[i].DataIn[j] := xArray[i]
    end;
  finally
    xTempStream.Free;
    xPatternsList.Free;
    SetLength(xArray, 0);
    xArray := nil;
  end;
end;

procedure TNeuralNetExtended.LoadPhase1;
begin
  FSourceFileName := FNnwFile.ReadString('Phase1', 'LearnSampleFileName', '');
  FNeuroDataSource.Name := FSourceFileName;
end;

procedure TNeuralNetExtended.LoadPhase2;
var
  i: integer;
begin
  AvailableFieldsCount := FNnwFile.ReadInteger('Phase2', 'AvailableFieldsCount',
1);
  for i := 0 to AvailableFieldsCount - 1 do
  with FFields[i] do
  begin
    Name := FNnwFile.ReadString('Phase2', 'FieldName_'+IntToStr(i), '');
    Kind := FNnwFile.ReadInteger('Phase2', 'FieldType_'+IntToStr(i), 0);
    NormType := FNnwFile.ReadInteger('Phase2', 'NormType_'+IntToStr(i), 0);
    ValueMax := FNnwFile.ReadFloat('Phase2', 'Max_'+IntToStr(i), 0);
    ValueMin := FNnwFile.ReadFloat('Phase2', 'Min_'+IntToStr(i), 0);
    ValueMid := FNnwFile.ReadFloat('Phase2', 'Mid_'+IntToStr(i), 0);
    Dispersion := FNnwFile.ReadFloat('Phase2', 'Disp_'+IntToStr(i), 0);
    Alpha := FNnwFile.ReadFloat('Phase2', 'Alpha_'+IntToStr(i), 0);
    Ind := FNnwFile.ReadBool('Phase2', 'Ind_'+IntToStr(i), False);
  end;
  SettingsLoaded := True;
end;

procedure TNeuralNetExtended.LoadPhase4;
begin
  UseForTeach := FNnwFile.ReadInteger('Phase4', 'UseForTeach',
DefaultUseForTeach);
  IdentError:= FNnwFile.ReadFloat('Phase4', 'IdentErr', DefaultErrorValue);
  TestAsValid := FNnwFile.ReadBool('Phase4', 'TestAsValid', False);
  Epoch:= FNnwFile.ReadBool('Phase4', 'Epoch', False);
  EpochCount:= FNnwFile.ReadInteger('Phase4', 'Epoch', DefaultEpochCount);
  MaxTeachError:= FNnwFile.ReadBool('Phase4', 'MaxTeachErr', False);
  MaxTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTeachErr',
DefaultErrorValue);
  MidTeachError:= FNnwFile.ReadBool('Phase4', 'MidTeachErr', False);
  MidTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTeachErr',
DefaultErrorValue);
  TeachIdent:= FNnwFile.ReadBool('Phase4', 'TeachIdent', False);
  TeachIdentCount:= FNnwFile.ReadInteger('Phase4', 'TeachIdent',
DefaultTeachIdentCount);
  MaxTestError:= FNnwFile.ReadBool('Phase4', 'MaxTestErr', False);
  MaxTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTestErr',
DefaultErrorValue);
  MidTestError:= FNnwFile.ReadBool('Phase4', 'MidTestErr', False);
  MidTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTestErr',
DefaultErrorValue);
  TestIdent:= FNnwFile.ReadBool('Phase4', 'TestIdent', False);
  TestIdentCount:= FNnwFile.ReadInteger('Phase4', 'TestIdent',
DefaultTestIdentCount);
end;

procedure TNeuralNetExtended.LoadNetwork;
var
```

```
  i,j,k: integer;
  xLayerCount: integer;
begin
  // знищується поточна конфігурація нейромереж системи кібербезпеки для
моделювання ботнет мережіі
  ResetLayers;
  Alpha := FNnwFile.ReadFloat('Network', 'Alpha', DefaultAlpha);
  Momentum := FNnwFile.ReadFloat('Network', 'Miu', DefaultMomentum);
  TeachRate := FNnwFile.ReadFloat('Network', 'TeachSpeed', DefaultTeachRate);
  EpochCount := FNnwFile.ReadInteger('Network', 'Epoch', DefaultEpochCount);
  xLayerCount := FNnwFile.ReadInteger('Network','CountLayers',
DefaultLayerCount);
  // задається кількість нейронів у шарах
  AutoInit := False;
  for i := 0 to xLayerCount - 1 do
    AddLayer(FNnwFile.ReadInteger('Network','Layer_'+IntToStr(i),
DefaultNeuronCount));
  AutoInit := True;
  // ініціалізація нової конфігурації нейромереж системи кібербезпеки для
моделювання ботнет мережіі
  Init;
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Weights[k] := FNnwFile.ReadFloat('Network',
                              'W_'+IntToStr( i-
1)+'_'+IntToStr(k)+'_'+IntToStr(j), 0);
      LayersBP[i].NeuronsBP[j].Weights[LayersBP[ i-1].NeuronCount] :=
FNnwFile.ReadFloat('Network',
       'WT_'+IntToStr( i-1)+'_'+IntToStr(j), 0);
    end;
end;

procedure TNeuralNetExtended.SavePhase1;
begin
  FNnwFile.WriteString('Phase1', 'LearnSampleFileName', FNeuroDataSource.Name);
end;

procedure TNeuralNetExtended.SavePhase2;
var
  i: integer;
begin
  FNnwFile.WriteInteger('Phase2', 'AvailableFieldsCount',
FAvailableFieldsCount);
  for i := 0 to AvailableFieldsCount - 1 do
  with Fields[i] do
  begin
    FNnwFile.WriteString('Phase2', 'FieldName_'+IntToStr(i), Name);
    FNnwFile.WriteInteger('Phase2', 'FieldType_'+IntToStr(i), Kind);
    FNnwFile.WriteInteger('Phase2', 'NormType_'+IntToStr(i), NormType);
    FNnwFile.WriteFloat('Phase2', 'Max_'+IntToStr(i), ValueMax);
    FNnwFile.WriteFloat('Phase2', 'Min_'+IntToStr(i), ValueMin);
    FNnwFile.WriteFloat('Phase2', 'Mid_'+IntToStr(i), ValueMid);
    FNnwFile.WriteFloat('Phase2', 'Disp_'+IntToStr(i), Dispersion);
    FNnwFile.WriteFloat('Phase2', 'Alpha_'+IntToStr(i), Alpha);
    FNnwFile.WriteBool('Phase2', 'Ind_'+IntToStr(i), Ind);
  end;
end;

procedure TNeuralNetExtended.SavePhase4;
begin
  FNnwFile.WriteBool('Phase4', 'Epoch', Epoch);
  FNnwFile.WriteInteger('Phase4', 'Epoch', EpochCount);
  FNnwFile.WriteFloat('Phase4', 'IdentErr', IdentError);
  FNnwFile.WriteBool('Phase4', 'MaxTeachErr', MaxTeachError);
  FNnwFile.WriteFloat('Phase4', 'MaxTeachErr', MaxTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MaxTestErr', MaxTestError);
```

```
  FNnwFile.WriteFloat('Phase4', 'MaxTestErr', MaxTestErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTeachErr', MidTeachError);
  FNnwFile.WriteFloat('Phase4', 'MidTeachErr', MidTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTestErr', MidTestError);
  FNnwFile.WriteFloat('Phase4', 'MidTestErr', MidTestErrorValue);
  FNnwFile.WriteFloat('Phase4', 'Miu', Momentum);
  FNnwFile.WriteBool('Phase4', 'TeachIdent', TeachIdent);
  FNnwFile.WriteFloat('Phase4', 'TeachSpeed', TeachRate);
  FNnwFile.WriteInteger('Phase4', 'TeachIdent', TeachIdentCount);
  FNnwFile.WriteBool('Phase4', 'TestAsValid', TestAsValid);
  FNnwFile.WriteBool('Phase4', 'TestIdent', TestIdent);
  FNnwFile.WriteInteger('Phase4', 'TestIdent', TestIdentCount);
  FNnwFile.WriteInteger('Phase4', 'UseForTeach', UseForTeach);
end;

procedure TNeuralNetExtended.SaveNetwork;
var
  i, j, k: integer;
begin
  FNnwFile.WriteFloat('Network', 'TeachSpeed', TeachRate);
  FNnwFile.WriteFloat('Network', 'Miu', Momentum);
  FNnwFile.WriteFloat('Network', 'Alpha', Alpha);
  FNnwFile.WriteInteger('Network', 'Epoch', EpochCount);
  FNnwFile.WriteInteger('Network','CountLayers', LayerCount);
  // задається кількість нейронів у шарах
  for i := 0 to LayerCount - 1 do
    FNnwFile.WriteInteger('Network','Layer_'+IntToStr(i),
StrToInt(NeuronsInLayer[i]));
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to StrToInt(NeuronsInLayer[i]) - 1 do
    begin
      for k := 0 to StrToInt(NeuronsInLayer[ i-1]) do
        FNnwFile.WriteFloat('Network','W_'+IntToStr( i-1)+'_'+IntToStr(k)+
                            '_'+IntToStr(j),
LayersBP[i].NeuronsBP[j].Weights[k]);
      FNnwFile.WriteFloat('Network','WT_'+IntToStr( i-1)+'_'+IntToStr(j),
         LayersBP[i].NeuronsBP[j].Weights[StrToInt(NeuronsInLayer[j])]);
   end;
end;

procedure TNeuralNetExtended.NormalizeData;
var
  i: integer;
begin
  // нормалізація вхідних і вихідних значень
  for i := 0 to FAvailableFieldsCount - 1 do
  begin
    FFields[i].FindMinMax;
    FFields[i].Normalize;
  end;
end;

procedure TNeuralNetExtended.Train;
var
  i, j, k: integer;
begin
  if FUseForTeach = 100 then
  begin
    PatternCount := FFields[0].DataInCount;
    TestSetPatternCount := 0;
  end
  else
  begin
    PatternCount := Round((FFields[0].DataInCount - 1) * FUseForTeach / 100);
    TestSetPatternCount := FFields[0].DataInCount - PatternCount;
  end;
  if not TeachStopped then
    NormalizeData;
```

```
  // формування вхідних значень навчальної множини
  RealOutputIndexCount := OutputFieldCount;
  RealInputIndexCount := InputFieldCount;
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsInput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealInputIndex[k] := i;
      Inc(k);
    end;
  // формування вихідних значень навчальної множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsOutput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealOutputIndex[k] := i;
      Inc(k);
    end;
  // формування вхідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatterns[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // формування вихідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatternsOut[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // навчання або донавчання мереж системи кібербезпеки для моделювання ботнет
мережіі
  TeachOffLine;
end;

procedure TNeuralNetExtended.SetAvailableFieldsCount(Value : integer);
var
  i: integer;
begin
  FAvailableFieldsCount := Value;
  // встановлюється кількість полів
  SetLength(FFields, Value);
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i] := TNeuroField.Create;
end;

procedure TNeuralNetExtended.SetFields(Index: integer; Value: TNeuroField);
begin
  try
    FFields[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SFieldIndexRange, [Index])
  end;
end;

procedure TNeuralNetExtended.SetDefaultProperties;
```

```
begin
  // параметри встановлювані за замовчуванням
  Epoch := False;
  IdentError:= DefaultErrorValue;
  MaxTeachError := False;
  MaxTeachErrorValue := DefaultErrorValue;
  MaxTestError:= False;
  MaxTestErrorValue:= DefaultErrorValue;
  MidTestError:= False;
  MidTestErrorValue:= DefaultErrorValue;
  MidTeachError := False;
  MidTeachErrorValue :=  DefaultErrorValue;
  SettingsLoaded := False;
  TeachIdent := False;
  TeachIdentCount:= DefaultTeachIdentCount;
  TestAsValid := False;
  TestIdent:= False;
  TestIdentCount:= DefaultTestIdentCount;
  UseForTeach := DefaultUseForTeach;
end;

procedure TNeuralNetExtended.SetFileName(Value: TFilename);
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  try
    FNnwFile := TIniFile.Create(Value);
    FFileName := Value;
  except
    on E: EInOutError do
      raise E.CreateFmt(SWrongFileName, [Value]);
  end;
  FNeuroDataSource := TNeuroDataSource.Create;
  LoadPhase1;
  LoadPhase2;
  LoadPhase4;
  LoadNetwork;
  LoadDataFrom;
end;

procedure TNeuralNetExtended.SetTeachIdentCount(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FTeachIdentCount := DefaultTeachIdentCount
  else
    FTeachIdentCount := Value;
end;

procedure TNeuralNetExtended.SetUseForTeach(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FUseForTeach := DefaultUseForTeach
  else
    FUseForTeach := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndex(Index: integer; const Value:
integer);
begin
  FRealOutputIndex[Index] := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndexCount(const Value: integer);
begin
  SetLength(FRealOutputIndex, Value)
end;
procedure TNeuralNetExtended.SetRealInputIndex(Index: integer; const Value:
integer);
begin
```

```
  FRealInputIndex[Index] := Value;
end;
procedure TNeuralNetExtended.SetRealInputIndexCount(const Value: integer);
begin
  SetLength(FRealInputIndex, Value)
end;
procedure Register;
begin
  RegisterComponents('BotnetNetwork', [TNeuralNetHopf, TNeuralNetBP,
TNeuralNetExtended]);
end;
end.
```

Pumpdata.pas – формування бази знань

```
unit PumpData;

interface

uses
  SysUtils, IniFiles, Classes, BotnetNetwork;

type

  EFieldNormError = class(Exception);
  EFieldKindError = class(Exception);

  TNeuroField = class;
  TNeuroFields = array of TNeuroField;

  TNeuroField = class(TObject)
  private
    FAlpha: double;
    FDataIn: TVectorFloat;
    FDispersion: double;
    FInd: boolean;
    FKind: byte;
    FName: string;
    FNormType: byte;
    FValueMax: double;
    FValueMid: double;
    FValueMin: double;
    function GetDataIn(Index: integer): double;
    function GetKindName: TNeuroFieldType;
    function GetNormTypeName: TNormalize;
    function GetDataInCount: integer;
    procedure SetDataIn(Index: integer; Value: double);
    procedure SetKind(Value: byte);
    procedure SetNormType(Value: byte);
    procedure SetDataInCount(Value: integer);
  public
    procedure FindMinMax;
    procedure CalcMid;
    procedure CalcDispersion;
    procedure Normalize;
    procedure DeNormalize;
    property Alpha: double read FAlpha write FAlpha;
    property DataIn[Index: integer]: double read GetDataIn write SetDataIn;
    property DataInCount: integer read GetDataInCount write SetDataInCount;
    property Dispersion: double read FDispersion write FDispersion;
    property Ind: boolean read FInd write FInd;
    property Kind: byte read FKind write SetKind;
    property KindName: TNeuroFieldType read GetKindName;
    property Name: string read FName write FName;
    property NormType: byte read FNormType write SetNormType;
    property NormTypeName: TNormalize read GetNormTypeName;
    property ValueMax: double read FValueMax write FValueMax;
    property ValueMin: double read FValueMin write FValueMin;
    property ValueMid: double read FValueMid write FValueMid;
  end;

  TNeuroDataSource = class(TObject)
  private
    FName: TFileName;
    function IsHeaderChar(AValue: char): boolean;
  public
    function FieldCount(AHeader: string): integer;
    procedure ExtractHeaders(const AFields: TNeuroFields; AHeader: string);
    procedure ExtractValues(const AVector:TVectorFloat; AHeader: string);
    property Name: TFileName read FName write FName;
  end;
```

```
implementation

{ Клас TNeuroField }

function TNeuroField.GetDataIn(Index: integer): double;
begin
  Result := FDataIn[Index];
end;

function TNeuroField.GetDataInCount: integer;
begin
  Result := High(FDataIn) + 1;
end;

function TNeuroField.GetKindName: TNeuroFieldType;
begin
  case FKind of
    0 : Result := fdInput;
    1 : Result := fdOutput;
    2 : Result := fdNone;
  end;
end;

function TNeuroField.GetNormTypeName: TNormalize;
begin
  case FNormType of
    0 : if KindName = fdInput then
          Result := nrmLinear
        else if KindName = fdOutput then
          Result := nrmLinearOut;
    1 : Result := nrmSigmoid;
    2 : Result := nrmAuto;
    3 : Result := nrmNone;
  end;
end;

procedure TNeuroField.CalcMid;
var
  i: integer;
begin
  FValueMid := 0;
  for i := Low(FDataIn) to High(FDataIn) do
    FValueMid := FValueMid + FDataIn[i];
  FValueMid := FValueMid/(High(FDataIn) + 1);
end;

procedure TNeuroField.CalcDispersion;
var
  i: integer;
begin
  if High(FDataIn) > 1 then
  begin
    FDispersion := 0;
    for i := Low(FDataIn) to High(FDataIn) do
      FDispersion := FDispersion + sqr(FDataIn[i] - ValueMid);
    FDispersion := sqrt(FDispersion/High(FDataIn));
  end
  else
    FDispersion := 0;
end;

(*procedure TNeuroField.DeNormalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
```

```
                FDataIn[i] := (FDataIn[i] + 1)*(FValueMax - FValueMin)/2 +
FValueMin;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := FDataIn[i]*(FValueMax - FValueMin) + FValueMin;
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := - Ln(1/FDataIn[i] - 1)/Alpha;
  end;
end;*)

procedure TNeuroField.FindMinMax;
var
  i: integer;
begin
  FValueMax:= FDataIn[0];
  FValueMin:= FDataIn[0];
  for i := 1 to High(FDataIn) do
  begin
    if FValueMin > FDataIn[i] then
      FValueMin := FDataIn[i];
    if FValueMax < FDataIn[i] then
      FValueMax := FDataIn[i]
  end;
end;

procedure TNeuroField.Normalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmAuto:  begin
                CalcMid;
                CalcDispersion;
                for i := Low(FDataIn) to High(FDataIn) do
                begin
                  xTmp := (FDataIn[i] - FValueMid)/FDispersion;
                  FDataIn[i] := 1/(1 + exp(-xTmp));
                end;
              end;
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 2*(FDataIn[i] - FValueMin)/(FValueMax - FValueMin)-
1;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := (FDataIn[i] - FValueMin)/(FValueMax - FValueMin);
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 1/( 1 + exp(-Alpha * FDataIn[i]));
  end;
end;

procedure TNeuroField.SetNormType(Value: byte);
begin
  if (Value < 0) or (Value > 3) then
    raise EFieldNormError.CreateFmt(SFieldNorm, [Value])
  else
    FNormType := Value;
end;

procedure TNeuroField.SetKind(Value: byte);
begin
  if (Value < 0) or (Value > 2) then
    raise Exception.CreateFmt(SFieldKind, [Value])
  else
    FKind := Value;
end;

procedure TNeuroField.SetDataIn(Index: integer; Value: double);
begin
  FDataIn[Index] := Value;
end;
```

```
procedure TNeuroField.SetDataInCount(Value: integer);
begin
  SetLength(FDataIn, Value)
end;

{ Клас TNeuroDataSource }

function TNeuroDataSource.IsHeaderChar(AValue: char): boolean;
begin
  if (AValue in Letters) or (AValue in Capitals) or (AValue in DigitChars) then
    Result := True
  else
    Result := False;
end;

procedure TNeuroDataSource.ExtractValues(const AVector:TVectorFloat; AHeader:
string);
var
  s: string;
  i, xCurPos: integer;
begin
  i := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  try
    while xCurPos > 0 do
    begin
      s := Copy(AHeader, 1, xCurPos - 1);
      AVector[i] := StrToFloat(s);
      Inc(i);
      Delete(AHeader, 1, xCurPos - 1);
      AHeader := Trim(AHeader);
      xCurPos := Pos(SpaceChar, AHeader);
    end;
    s := AHeader;
    AVector[i] := StrToFloat(s);
  except
    on EConvertError do
      EConvertError.CreateFmt(SCannotBeNumber, [s])
  end;
end;

procedure TNeuroDataSource.ExtractHeaders(const AFields: TNeuroFields; AHeader:
string);
var
  s: string;
  xFieldCount, j, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    s := Copy(AHeader, 1, xCurPos - 1);
    AFields[xFieldCount].FName := '';
    for j := 1 to Length(s) do
      if isHeaderChar(s[j]) then
        AFields[xFieldCount].FName := AFields[xFieldCount].FName + s[j];
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  AFields[xFieldCount].FName := '';
  for j := 1 to Length(AHeader) do
    if isHeaderChar(AHeader[j]) then
      AFields[xFieldCount].FName := AFields[xFieldCount].FName + AHeader[j];
```

```
  { повертає кількість полів }
end;

function TNeuroDataSource.FieldCount(AHeader: string): integer;
var
  xFieldCount, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  { повертає кількість полів }
  Result := xFieldCount + 1;
end;


end.
```

NeuralNetExtend.pas – навчання мереж системи кібербезпеки для моделювання ботнет мережіі

```
unit NeuralNetExtend;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, BotnetNetworkComp, ExtCtrls, StdCtrls, Spin, Grids, BotnetNetwork,
  IniFiles;

const
  FormCaption = 'Навчання мереж системи кібербезпеки для моделювання ботнет
мережіі';

type
  TfrmNeuralNetExtend = class(TForm)
    PageControl: TPageControl;
    pnlNavigation: TPanel;
    Tab1: TTabSheet;
    btnBack: TButton;
    rgrFileType: TRadioGroup;
    btnNext: TButton;
    btnCancel: TButton;
    Tab2: TTabSheet;
    lblFileName: TLabel;
    btnOpenFile: TButton;
    edtFileName: TEdit;
    OpenDialog: TOpenDialog;
    Tab3: TTabSheet;
    ltbFieldName: TListBox;
    Label2: TLabel;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    edtMin: TEdit;
    Label4: TLabel;
    edtMax: TEdit;
    Label5: TLabel;
    edt: TEdit;
    Tab4: TTabSheet;
    speLayers: TSpinEdit;
    Label6: TLabel;
    stgNeuronsInLayer: TStringGrid;
    Label7: TLabel;
    Tab5: TTabSheet;
    Label8: TLabel;
    tbrAlpha: TTrackBar;
    sttAlpha: TStaticText;
    Label9: TLabel;
    edtMomentum: TEdit;
    Label10: TLabel;
    edtTeachRate: TEdit;
    Tab6: TTabSheet;
    Label11: TLabel;
    Label12: TLabel;
    btnContinueTeach: TButton;
    sttMaxTeachError: TStaticText;
    sttEpochCount: TStaticText;
    GroupBox2: TGroupBox;
    Label13: TLabel;
    edtIdentError: TEdit;
    speEpochCount: TSpinEdit;
    cbxEpoch: TCheckBox;
    cbxMaxTeachError: TCheckBox;
```

```
edtMaxTeachErrorValue: TEdit;
cbxMidTeachError: TCheckBox;
edtMidTeachErrorValue: TEdit;
cbxTeachIdent: TCheckBox;
speTeachIdentValue: TSpinEdit;
btnBeginTeach: TButton;
cbxMaxTestError: TCheckBox;
cbxMidTestError: TCheckBox;
cbxTestIdent: TCheckBox;
edtMaxTestErrorValue: TEdit;
edtMidTestErrorValue: TEdit;
speTestIdentValue: TSpinEdit;
Tab7: TTabSheet;
Label14: TLabel;
stgInput: TStringGrid;
Label15: TLabel;
stgOutput: TStringGrid;
btnCompute: TButton;
Memo1: TMemo;
Label16: TLabel;
Label17: TLabel;
Memo2: TMemo;
Bevel1: TBevel;
Memo3: TMemo;
Label1: TLabel;
sttMaxTestError: TStaticText;
Label18: TLabel;
sttMidTeachError: TStaticText;
Label19: TLabel;
sttMidTestError: TStaticText;
SaveDialog: TSaveDialog;
edtUseForTeach: TEdit;
Label20: TLabel;
btnSave: TButton;
procedure btnCancelClick(Sender: TObject);
procedure btnNextClick(Sender: TObject);
procedure btnBackClick(Sender: TObject);
procedure btnOpenFileClick(Sender: TObject);
procedure FormActivate(Sender: TObject);
procedure ltbFieldNameClick(Sender: TObject);
procedure rdgFieldTypeClick(Sender: TObject);
procedure rdgNormTypeClick(Sender: TObject);
procedure edtAChange(Sender: TObject);
procedure tbrAlphaChange(Sender: TObject);
procedure edtMomentumChange(Sender: TObject);
procedure edtTeachRateChange(Sender: TObject);
procedure NeuralNetExtendedEpochPassed(Sender: TObject);
procedure btnContinueTeachClick(Sender: TObject);
procedure edtIdentErrorChange(Sender: TObject);
procedure cbxEpochClick(Sender: TObject);
procedure speEpochCountChange(Sender: TObject);
procedure cbxMaxTeachErrorClick(Sender: TObject);
procedure edtMaxTeachErrorValueChange(Sender: TObject);
procedure cbxMidTeachErrorClick(Sender: TObject);
procedure edtMidTeachErrorValueChange(Sender: TObject);
procedure cbxTeachIdentClick(Sender: TObject);
procedure speTeachIdentValueChange(Sender: TObject);
procedure cbxMaxTestErrorClick(Sender: TObject);
procedure edtMaxTestErrorValueChange(Sender: TObject);
procedure cbxMidTestErrorClick(Sender: TObject);
procedure edtMidTestErrorValueChange(Sender: TObject);
procedure cbxTestIdentClick(Sender: TObject);
procedure speTestIdentValueChange(Sender: TObject);
procedure NeuralNetExtendedAfterTeach(Sender: TObject);
procedure btnBeginTeachClick(Sender: TObject);
procedure btnComputeClick(Sender: TObject);
procedure speLayersChange(Sender: TObject);
procedure btnSaveClick(Sender: TObject);
procedure edtUseForTeachChange(Sender: TObject);
```

```
  private
    { Private declarations }
    Teach: boolean;
    NotSaved: boolean;
    procedure ChangePage;
    procedure OpenFile;
    procedure LoadFile;
    procedure Tune;
    procedure CreateNet;
    procedure RunTeach;
    procedure TestNet;
  public
    { Public declarations }
  end;

var
  frmNeuralNetExtend: TfrmNeuralNetExtend;
implementation

{$R *.DFM}

// Запуск програми
procedure TfrmNeuralNetExtend.FormActivate(Sender: TObject);
begin
  Caption := FormCaption;
  PageControl.ActivePage := PageControl.Pages[0];
  Teach := false;
  NotSaved := false;
end;

// Вихід із програми
procedure TfrmNeuralNetExtend.btnCancelClick(Sender: TObject);
begin
  if NotSaved then
    if MessageDlg('Є незбережені дані. Зберегти?',mtConfirmation, [mbYes, mbNo],
0) = mrYes then
      btnSave.Click;
  Close;
end;

// Натискання кнопки "Вперед"
procedure TfrmNeuralNetExtend.btnNextClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, true, false);
    ChangePage;
    if ActivePage.PageIndex = PageCount - 1 then
      btnNext.Enabled := false
    else
      btnNext.Enabled := true;
    btnBack.Enabled := true;
  end;
end;

// Натискання кнопки "Назад"
procedure TfrmNeuralNetExtend.btnBackClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, false, false);
    if ActivePage.PageIndex = 0 then
      btnBack.Enabled := false
    else
      btnBack.Enabled := true;
    btnNext.Enabled := true;
  end;
end;
```

```
// Дія на зміну сторінки
procedure TfrmNeuralNetExtend.ChangePage;
begin
  case PageControl.ActivePage.PageIndex of
    1: OpenFile;
    2: LoadFile;
    3: Tune;
    4: CreateNet;
    6: TestNet;
  end;
end;

// Вибрати файл - джерело даних
procedure TfrmNeuralNetExtend.OpenFile;
begin
  if rgrFileType.ItemIndex = 0 then
  begin
    OpenDialog.Filter := 'NNW files (*.nnw)|*.nnw';
    lblFileName.Caption := 'Виберіть nnw-файл';
  end
  else
  begin
    OpenDialog.Filter := 'Text files (*.txt)|*.txt';
    lblFileName.Caption := 'Виберіть txt-файл';
  end;
end;

// Вибір файлу в діалоговому вікні
procedure TfrmNeuralNetExtend.btnOpenFileClick(Sender: TObject);
begin
  OpenDialog.Execute;
  Caption := FormCaption + ' - ' + ExtractFileName(OpenDialog.FileName);
  edtFileName.Text := OpenDialog.FileName;
end;

// Завантажити в компонент обраний файл
procedure TfrmNeuralNetExtend.LoadFile;
var
  i: integer;
begin
  try
    if rgrFileType.ItemIndex = 0 then
      NeuralNetExtended.FileName := edtFileName.Text   // nnw-файл
    else
    begin
      NeuralNetExtended.SourceFileName := edtFileName.Text; // текстовий файл
      NeuralNetExtended.LoadDataFrom; // завантажує дані з текстового файлу
      // конфігурація нейронної мереж системи кібербезпеки для моделювання
ботнет мережіі за замовчуванням
      NeuralNetExtended.AddLayer(2);
      NeuralNetExtended.AddLayer(3);
      NeuralNetExtended.AddLayer(1);
    end;
  except
    raise Exception.Create('Помилка при відкритті файлу');
  end;
  NeuralNetExtended.Init;  // Ініціалізація мереж системи кібербезпеки для
моделювання ботнет мережіі
  // Формування списку полів для StringList-A
  ltbFieldName.Clear;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    ltbFieldName.Items.Add(NeuralNetExtended.Fields[i].Name);
  ltbFieldName.ItemIndex := 0;
  ltbFieldNameClick(Self);
end;

// Настроювання параметрів мереж системи кібербезпеки для моделювання ботнет
мережіі
procedure TfrmNeuralNetExtend.Tune;
```

```
var
  i: integer;
begin
  speLayers.Value := NeuralNetExtended.LayerCount - 2;
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  stgNeuronsInLayer.Cells[0, 0] := '№ шаруючи';
  stgNeuronsInLayer.Cells[1, 0] := 'Нейронів';
  for i := 0 to speLayers.Value - 1 do
  begin
    stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
    stgNeuronsInLayer.Cells[1, i + 1] := IntToStr(NeuralNetExtended.Layers[i +
1].NeuronCount);
  end;

  tbrAlpha.Position := trunc(NeuralNetExtended.Alpha * 100);
  edtMomentum.Text := FloatToStr(NeuralNetExtended.Momentum);
  edtTeachRate.Text := FloatToStr(NeuralNetExtended.TeachRate);
  edtIdentError.Text := FloatToStr(NeuralNetExtended.IdentError);
  edtUseForTeach.Text := FloatToStr(NeuralNetExtended.UseForTeach);

  cbxEpoch.Checked := NeuralNetExtended.Epoch;
  speEpochCount.Text := IntToStr(NeuralNetExtended.EpochCount);

  cbxMaxTeachError.Checked := NeuralNetExtended.MaxTeachError;
  edtMaxTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MaxTeachErrorValue);

  cbxMidTeachError.Checked := NeuralNetExtended.MidTeachError;
  edtMidTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MidTeachErrorValue);

  cbxTeachIdent.Checked := NeuralNetExtended.TeachIdent;
  speTeachIdentValue.Value := NeuralNetExtended.TeachIdentCount;

  cbxMaxTestError.Checked := NeuralNetExtended.MaxTestError;
  edtMaxTestErrorValue.Text := FloatToStr(NeuralNetExtended.MaxTestErrorValue);

  cbxMidTestError.Checked := NeuralNetExtended.MidTestError;
  edtMidTestErrorValue.Text := FloatToStr(NeuralNetExtended.MidTestErrorValue);

  cbxTestIdent.Checked := NeuralNetExtended.TestIdent;
  speTestIdentValue.Value := NeuralNetExtended.TeachIdentCount;
end;

// Відображення інформації про обране поле (тип поля, нормалізація та інше)
procedure TfrmNeuralNetExtend.ltbFieldNameClick(Sender: TObject);
begin
  // Тип поля - вхідне, вихідне, не використовувати
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  // Тип нормалізації
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  // Параметр нормалізації
  edt.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  // Мінінум та максимум (для лінійної нормалізації)
  edtMin.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  edtMax.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

// Змінити тип поля
procedure TfrmNeuralNetExtend.rdgFieldTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
end;
```

```
// Змінити тип нормалізації
procedure TfrmNeuralNetExtend.rdgNormTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
end;

// Змінити параметр нормалізації
procedure TfrmNeuralNetExtend.edtAChange(Sender: TObject);
begin
 NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha := StrToFloat(edt.Text);
end;

// Змінити параметр Alpha мереж системи кібербезпеки для моделювання ботнет
мережіі
procedure TfrmNeuralNetExtend.tbrAlphaChange(Sender: TObject);
begin
  sttAlpha.Caption := FloatToStr(tbrAlpha.Position / 100);
end;

// Зміна кількості схованих шарів
procedure TfrmNeuralNetExtend.speLayersChange(Sender: TObject);
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
end;

// Створення мереж системи кібербезпеки для моделювання ботнет мережіі обраної
топологіі
procedure TfrmNeuralNetExtend.CreateNet;
var
  i: integer;
  xInput, xOutput: integer;
begin
  // Змінюється тільки кількість нейронів у схованих шарах,
  // Кількість нейронів у вхідному й вихідному шарі залежить від
  // типів полів
  with NeuralNetExtended do
  begin
    xInput := InputFieldCount;
    xOutput := OutputFieldCount;
    ResetLayers;
    AddLayer(xInput);
    for i := 0 to speLayers.Value - 1 do
      AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i + 1]));
    AddLayer(xOutput);
  end;
end;

// Змінити момент мереж системи кібербезпеки для моделювання ботнет мережіі
procedure TfrmNeuralNetExtend.edtMomentumChange(Sender: TObject);
begin
  NeuralNetExtended.Momentum := StrToFloat(edtMomentum.Text);
end;

// Змінити швидкість навчання мереж системи кібербезпеки для моделювання ботнет
мережіі
procedure TfrmNeuralNetExtend.edtTeachRateChange(Sender: TObject);
begin
  NeuralNetExtended.TeachRate := StrToFloat(edtTeachRate.Text);
end;

// Дія по проходженню однієї епохи навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedEpochPassed(Sender: TObject);
begin
  sttMaxTestError.Caption := '';
  sttMidTestError.Caption := '';
  with NeuralNetExtended do
  begin
```

```
    sttMaxTeachError.Caption := FloatToStr(MaxTeachResidual); // Показати макс.
помилку на навчальній множині
    sttMidTeachError.Caption := FloatToStr(MidTeachResidual); // Показати серед.
помилку на навчальній множині
    if NeuralNetExtended.UseForTeach <> 100 then
    begin
      sttMaxTestError.Caption := FloatToStr(MaxTestResidual); // Показати макс.
помилку на тестовій множині
      sttMidTestError.Caption := FloatToStr(MidTestResidual); // Показати сред.
помилку на тестовій множині
    end;
    sttEpochCount.Caption := FloatToStr(EpochCurrent); // Показати номер
поточної епохи
  end;
  Application.ProcessMessages; // Дати можливість Windows перемалювати форму
end;

// Натискання кнопки "Продовжити навчання"
procedure TfrmNeuralNetExtend.btnContinueTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := true; // Скинути прапор - "Продовжити
навчання"
  RunTeach;  // Запуск
end;

// Натискання кнопки "Навчити"
procedure TfrmNeuralNetExtend.btnBeginTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := false; // Скинути прапор - "Почати навчання
знову"
  RunTeach;
end;

procedure TfrmNeuralNetExtend.edtIdentErrorChange(Sender: TObject);
begin
  NeuralNetExtended.IdentError := StrToFloat(edtIdentError.Text);
end;

procedure TfrmNeuralNetExtend.edtUseForTeachChange(Sender: TObject);
begin
  NeuralNetExtended.UseForTeach := StrToInt(edtUseForTeach.Text);
end;

procedure TfrmNeuralNetExtend.cbxEpochClick(Sender: TObject);
begin
  NeuralNetExtended.Epoch := cbxEpoch.Checked;
end;

procedure TfrmNeuralNetExtend.speEpochCountChange(Sender: TObject);
begin
  NeuralNetExtended.EpochCount := StrToInt(speEpochCount.Text);
end;

procedure TfrmNeuralNetExtend.cbxMaxTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachError := cbxMaxTeachError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachErrorValue :=
StrToFloat(edtMaxTeachErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTeachError := cbxMidTeachError.Checked;
end;
```

```
procedure TfrmNeuralNetExtend.edtMidTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTeachErrorValue :=
StrToFloat(edtMidTeachErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxTeachIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TeachIdent := cbxTeachIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTeachIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTeachIdentValue.Value;
end;

procedure TfrmNeuralNetExtend.cbxMaxTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTestError := cbxMaxTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTestErrorValue := StrToFloat(edtMaxTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTestError := cbxMidTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMidTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTestErrorValue := StrToFloat(edtMidTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxTestIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TestIdent := cbxTestIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTestIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTestIdentValue.Value;
end;

// Зупинка навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedAfterTeach(Sender: TObject);
begin
  btnBack.Enabled := true;
  btnNext.Enabled := true;
  btnCancel.Enabled := true;
  btnContinueTeach.Caption := 'Навчити';
  NeuralNetExtended.StopTeach := true;
end;

procedure TfrmNeuralNetExtend.RunTeach;
begin
  Teach := not Teach;  // Перемикач стану "вчимося/не вчимося"
  if Teach then
  begin
    btnBeginTeach.Enabled := false;
    btnBack.Enabled := false;
    btnNext.Enabled := false;
    btnCancel.Enabled := false;
    NeuralNetExtended.StopTeach := false;
    btnContinueTeach.Caption := 'Зупинити навчання';
    NotSaved := true;
```

```
    NeuralNetExtended.Train;  // Запуск нейромереж системи кібербезпеки для
моделювання ботнет мережіі на навчання
    btnCompute.Enabled := true;
    btnSave.Enabled := true;
  end
  else
  begin
    btnBeginTeach.Enabled := true;
    btnBack.Enabled := true;
    btnNext.Enabled := true;
    btnCancel.Enabled := true;
    btnContinueTeach.Caption := 'Продовжити навчання';
    NeuralNetExtended.StopTeach := true; // Зупинити навчання
  end;
end;

// Відкриття сторінки - тестування навченої нейромереж системи кібербезпеки для
моделювання ботнет мережіі
procedure TfrmNeuralNetExtend.TestNet;
var
  i, j: integer;
begin
  stgInput.RowCount := NeuralNetExtended.InputFieldCount + 1;
  stgInput.Cells[0, 0] := 'Поле';
  stgInput.Cells[1, 0] := 'Значення';

  // Проставити імена вхідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdInput) then // Ознака того, що
поле вхідне
    begin
      Inc(j);
      stgInput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
  stgOutput.RowCount := NeuralNetExtended.OutputFieldCount + 1;
  stgOutput.Cells[0, 0] := 'Поле';
  stgOutput.Cells[1, 0] := 'Значення';

  // Проставити імена вихідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdOutput) then  // Ознака того,
що поле вихідне
    begin
      Inc(j);
      stgOutput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
end;

// Натискання кнопки "Обчислити"
procedure TfrmNeuralNetExtend.btnComputeClick(Sender: TObject);
var
  xVectorFloat: TVectorFloat;
  i: integer;
begin
  // Створити вектор, що будемо подавати на вхід
  // довжиною, рівною кількості нейронів на вхідному шарі
  SetLength(xVectorFloat, NeuralNetExtended.InputFieldCount);
  // Заповнити значення елементів вектора
  for i := 0 to NeuralNetExtended.InputFieldCount - 1 do
    xVectorFloat[i] := StrToFloat(stgInput.Cells[1, i + 1]);
  // Подати на вхід нейромереж системи кібербезпеки для моделювання ботнет
мережіі. Результати будуть у вихідному шарі нейромереж системи кібербезпеки для
моделювання ботнет мережіі
  NeuralNetExtended.ComputeUnPrepData(xVectorFloat);
  // Відобразити отримані результати
  for i := 0 to NeuralNetExtended.OutputFieldCount - 1 do
    stgOutput.Cells[1, i + 1] := FloatToStr(NeuralNetExtended.Output[i]);
```

```
  // Знищити вектор
  SetLength(xVectorFloat, 0);
  xVectorFloat := nil;
end;

// Зберегти навчену нейромереж системи кібербезпеки для моделювання ботнет
мережіу
procedure TfrmNeuralNetExtend.btnSaveClick(Sender: TObject);
begin
  SaveDialog.InitialDir := ExtractFilePath(NeuralNetExtended.FileName);
  SaveDialog.FileName := ExtractFileName(NeuralNetExtended.FileName);
  if SaveDialog.Execute then
  begin
    NeuralNetExtended.NnwFile := TIniFile.Create(SaveDialog.FileName);
    NeuralNetExtended.SavePhase1;
    NeuralNetExtended.SavePhase2;
    NeuralNetExtended.SavePhase4;
    NeuralNetExtended.SaveNetwork;
    NotSaved := false;
  end;
end;

end.
```

Hopf.pas - Блок формування та розпізнавання образів

```
unit Hopf;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, BotnetNetworkComp, BotnetNetwork, Db, DBTables, ExtCtrls, DBCtrls,
StdCtrls,
  ToolWin, ComCtrls;

type
  TForm1 = class(TForm)
    Table: TTable;
    btnExecute: TButton;
    DBNavigator: TDBNavigator;
    DataSource: TDataSource;
    btnEdit: TButton;
    stgDatabase: TStringGrid;
    stgInput: TStringGrid;
    stgOutput: TStringGrid;
    NeuralNetHopf: TNeuralNetHopf;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    Bevel: TBevel;
    TableLETTERS: TStringField;
    dbMain: TDatabase;
    procedure DataSourceDataChange(Sender: TObject; Field: TField);
    procedure FormActivate(Sender: TObject);
    procedure GridClick(Sender: TObject);
    procedure btnEditClick(Sender: TObject);
    procedure btnExecuteClick(Sender: TObject);
    procedure GridDrawCell(Sender: TObject; ACol, ARow: Integer;
      Rect: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
  public
    { Public declarations }
    procedure AddPattern(Value: string);
    procedure Clear(Grid: TStringGrid);
    procedure Init;
    procedure ShowMatrix(Grid: TStringGrid; Value: string);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

// Показати вектор у вигляді сітки
procedure TForm1.ShowMatrix(Grid: TStringGrid; Value: string);
var
  i, j: integer;
begin
  Clear(Grid);
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
    begin
      try
        if Value[i * Grid.RowCount + j + 1] = '1' then
          Grid.Cells[i, j] := '1'
        else
          Grid.Cells[i, j] := ' '
      except
        Grid.Cells[i, j] := ' '
```

```
      end;
    end;
end;

// Очистити сітку
procedure TForm1.Clear(Grid: TStringGrid);
var
  i, j: integer;
begin
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
      Grid.Cells[i, j] := ' ';
end;

// Показати символ з таблиці
procedure TForm1.DataSourceDataChange(Sender: TObject; Field: TField);
begin
  ShowMatrix(stgDatabase, TableLETTERS.Value);
end;

// Ініціалізація мереж системи кібербезпеки для моделювання ботнет мережіі
значеннями з таблиці
procedure TForm1.Init;
begin
  // Очистити мереж системи кібербезпеки для моделювання ботнет мережіу від
зразків
  NeuralNetHopf.ResetPatterns;

  // Додати зразки з таблиці до мереж системи кібербезпеки для моделювання
ботнет мережіі
  Table.First;
  while not Table.Eof do
  begin
    AddPattern(TableLETTERS.AsString);
    Table.Next;
  end;
  Table.First;

  // Ініціалізувати ваги
  NeuralNetHopf.InitWeights;
  // Мереж системи кібербезпеки для моделювання ботнет мережіа підготовлена до
розпізнавання
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  Clear(stgDatabase);
  Clear(stgInput);
  Clear(stgOutput);
  Init;
end;

procedure TForm1.GridClick(Sender: TObject);
begin
  with Sender as TStringGrid do
    if Cells[Col, Row] = '1' then
      Cells[Col, Row] := ' '
    else
      Cells[Col, Row] := '1'
end;

// Додавання нового образа до мереж системи кібербезпеки для моделювання ботнет
мережіі
procedure TForm1.AddPattern(Value: string);
var
  i: integer;
  xVector: TVectorInt;
begin
  SetLength(xVector, stgDatabase.RowCount * stgDatabase.ColCount);
```

```
  // Перетворення символьного рядка у вектор
  for i := 1 to stgDatabase.RowCount * stgDatabase.ColCount do
    try
      if TableLETTERS.AsString[i] = '1' then
        xVector[i - 1] := 1
      else
        xVector[i - 1] := -1;
    except
      xVector[i - 1] := -1;
    end;


  NeuralNetHopf.AddPattern(xVector);
end;

procedure TForm1.btnEditClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  xString := '';
  for i := 0 to stgDatabase.ColCount - 1 do
    for j := 0 to stgDatabase.RowCount - 1 do
      if stgDatabase.Cells[i, j] = '1' then
        xString := xString + '1'
      else
        xString := xString + ' ';
  Table.Edit;
  TableLETTERS.AsString := xString;
  Table.Post;
end;

// Розпізнавання символу
procedure TForm1.btnExecuteClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  // Подаємо сигнали на вихід мереж системи кібербезпеки для моделювання ботнет
мережіі
  for i := 0 to stgInput.ColCount - 1 do
    for j := 0 to stgInput.RowCount - 1 do
      if stgInput.Cells[i, j] = '1' then
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := 1
      else
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := -1;

  // Запуск процесу розпізнавання
  NeuralNetHopf.Calc;

  // Перетворення виходів мереж системи кібербезпеки для моделювання ботнет
мережіі до рядка
  xString := '';
  for i := 1 to stgOutput.RowCount * stgOutput.ColCount do
    if NeuralNetHopf.Layers[1].Neurons[i - 1].Output = 1 then
      xString := xString + '1'
    else
      xString := xString + ' ';

  // Відобразити результат
  ShowMatrix(stgOutput, xString);
end;

procedure TForm1.GridDrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  with Sender as TStringGrid do
  begin
```

```
    Canvas.Brush.Color := clBlack;
    if Cells[ACol,ARow] <> ' ' then
      Canvas.FillRect(Rect)
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  dbMain.Params.Values['Path'] := ExtractFilePath(Application.ExeName);
  dbMain.Open;
  Table.Open;

end;

end.
```