Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

ДИПЛОМНА РОБОТА МАГІСТРА

Розподілена самоорганізована система прогнозування зловмисної активності в комп'ютерних мережах
Назва теми

Галузь знань 12 – Інформаційні технології

Спеціальність 123 –Комп'ютерна інженерія

ДРКІ. 170146.21.01.06 ПЗ

Виконав: студент 2 курсу, групи КІ1м-21-1 _____ Любінецький Д.В.
Підпис

Керівник д. т. н, проф. _____ Савенко О.С.
Підпис

Нормоконтролер ст. викладач кафедри КБ _____ Мостовий С.В.
Підпис

До захисту допускаю:
Зав. кафедри КБ к.т.н., доц _____ Кльоц Ю.П.
Підпис

07 12 2022р.

Хмельницький, 2022

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра КІБЕРБЕЗПЕКИ
Освітній рівень МАГІСТР
Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ
Спеціальність 123 КОМП`ЮТЕРНА ІНЖЕНЕРІЯ
Освітня програма ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА ПІДГОТОВКИ МАГІСТРА

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
к.т.н. доцент Кльоц Ю.П.
" 1 " 09 2022 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ)

Любінецькому Денису Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розподілена самоорганізована система прогнозування зловмисної активності в комп`ютерних мережах.

Науковий керівник Савенко Олег Станіславович, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом № ректора університету додаток № від . .2022
2. Строк подання студентом проекту (роботи) на кафедру . .2022.
3. Вихідні дані до проекту (роботи) Дослідження розподілених самоорганізованих систем для прогозування.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз систем прогнозування зловмисної активності в комп`ютерних мережах, Аналіз джерел зловмисної активності, Дослідження розподілених систем виявлення зловмисного програмного забезпечення, Аналіз засобів використання штучного інтелекту для задач прогнозування, Формалізація зловмисного програмного забезпечення, класифікація проявів зловмисного програмного забезпечення, побудова системи самоорганізованої розподіленої структури, Програмна реалізаці самоорганізованої системи виявлення зловмисної активності.
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень):
Архітектура розподіленої самоорганізованої системи
Інформаційні потоки системи прогнозування

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Відповідальний за оформлення ДП	Мостовий С.В., ст викладач		

7. Дата видачі завдання: « 1 » лютого 2022 р

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Грунтовне ознайомлення з предметною галуззю	2.02.2022	Виконано
2	Визначення структури магістерської роботи	2.03. 2022	Виконано
3	Робота над першим розділом магістерської роботи	1.04. 2022	Виконано
4	Робота над першою статтею за результатами обробки літературних джерел	1.05. 2022	Виконано
5	Робота над другим розділом магістерської роботи	1.06. 2022	Виконано
6	Робота над третім розділом магістерської роботи	1.09. 2022	Виконано
7	Робота над четвертим розділом магістерської роботи	1.10. 2022	Виконано
8	Підготовка ілюстративного матеріалу	1.11. 2022	Виконано
9	Оформлення текстової і графічної частини магістерської роботи	11.11. 2022	Виконано
10	Попередній захист магістерської роботи	25.11. 2022	Виконано
11	Захист ДРМ на засіданні ЕК	8.12. 2022	Виконано

Студент _____ Д.В. Любінецький
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О.С. Савенко
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема магістерської роботи: «Розподілена самоорганізована система прогнозування зловмисної активності в комп’ютерних мережах».

Автор роботи: Любінецький Денис Володимирович.

Керівник роботи: Савенко Олег Станіславович.

Пояснювальна записка: 77 с., 25 рис., 3 табл., 3 дод., 83 джерел.

Графічна частина: 2 плакати.

Метою кваліфікаційної роботи: створити розподілену самоогранізовану систему виявлення зловмисної активності комп’ютерних мереж дослідити вплив комп’ютерних загроз та методи боротьби з ними, перевірити ефективність запропонованого рішення.

У магістерській роботі було проаналізовано системи прогнозування зловмисної активності, розподілені системи виявлення зловмисного програмного забезпечення, формалізував та класифікував прояви зловмисної активності в комп’ютерних мережах, дослідив архітектури самоорганізованих систем, їх проблеми та переваги. Після того спроектував систему на основі досліджених самоорганізованих алгоритмів. Крім того модифікував один із алгоритмів нейронної мережі, а саме алгоритм самоорганізованої інкрементної нейронної мережі для підвищення ефективності аналізу вхідного мережевого трафіку системи. Реалізував даний метод з допомогою мови програмування Python та методів штучного інтелекту.

В завершенні роботи я проаналізував запропоноване рішення та довів його ефективність. Опираючись на викладені моменти, я вважаю, що поставлена задача була виконана в повній мірі.

Підпис студента

5.12.22

Дата

ANNOTATION

The topic of the master's thesis: "Distributed self-organized system for predicting malicious activity in computer networks."

Author of the work: Liubinetskyi Denys Volodymyrovych.

Head of work: Savenko Oleg Stanislavovych.

Explanatory note: 77 pages, 25 figures, 3 tables, 3 appendices, 83 sources.

Graphic part: 2 posters.

The purpose of the qualification work: to create a distributed self-limited system for detecting malicious activity of computer networks, to investigate the impact of computer threats and methods of combating them, to check the effectiveness of the proposed solution.

The master's thesis analyzed malicious activity prediction systems, distributed malware detection systems, formalized and classified manifestations of malicious activity in computer networks, investigated the architecture of self-organized systems, their problems and advantages. After that, he designed a system based on researched self-organizing algorithms. In addition, he modified one of the algorithms of the neural network, namely the algorithm of the self-organized incremental neural network to increase the efficiency of the analysis of the incoming network traffic of the system. Implemented this method using the Python programming language and artificial intelligence methods.

At the end of the work, I analyzed the proposed solution and proved its effectiveness. Based on the points presented, I believe that the task was fully accomplished.

Student Signature

5.12.22

Date

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ANN	– artificial neural networks
API	– application programming interface
DIDS	– distributed instruction detection system
DL	– deep learning
IDS	– instruction detection system
MLP	– multi-layer perceptron
ЗПЗ	– зловмисне програмне забезпечення
АЗ	– апаратне забезпечення
ОС	– операційна система
ПЗ	– програмне забезпечення
ЦП	– центральний процесор
ШНМ	– штучна нейронна мережа
ШІ	– штучний інтелект

ВСТУП

Виявлення мережевих атак на теперішній момент є однією з найбільш вагомою проблемою у галузі мережевих технологій. Однією з актуальних наукових завдань в даний час є аналіз та прогнозування які використовують системи аналізу мережевого трафіку в сучасних мережах. Для вирішення завдання необхідний збір та аналіз різноманітної статистики в діючих мережах.

Метою та завданням магістерської роботи є створення розподіленої самоорганізованої системи прогнозування зловмисної активності в комп’ютерних мережах.

Об'єктом дослідження є інтелектуальні технології які базовані на нейромеревих принципах.

Задача дослідження полягає у створенні розподіленої самоорганізованої системи прогнозування зловмисної активності.

Реалізація поставленої задачі дозволить визначити, спроектувати та реалізувати розподілену систему прогнозуання зловмисної активності шляхом використання нейронних мереж та методів алгоритмів самоорганізації. Крім того, така система буде використовувати компоненти розподілені між кількома комп’ютерами у мережі, що дозволить їй працювати навіть після виходу з ладу частини вузлів.

1 АНАЛІЗ СИСТЕМ ПРОГНОЗУВАННЯ ЗЛОВМИСНОЇ АКТИВНОСТІ В КОМП'ЮТЕРНИХ МЕРЕЖАХ

1.1 Аналіз джерел зловмисної активності, їх наслідки та методи виявлення

Мережеві технології дозволяють обмінюватися, доставляти та отримувати інформацію з будь-якої точки земної кулі. Незважаючи на величезні переваги комп'ютерного обміну даними, інформація стає сприйнятливою до зловмисної діяльності, що завдає мільярдів доларів збитків бізнесу, урядам і промисловості. З'явився новий термін в комп'ютерній термінології – хакер [1]. Це загальний термін для опису людей, які вміють маніпулювати та атакувати комп'ютерні мережі. Масштаб проблеми, викликаної хакерською діяльністю, можна розкрити, проаналізувавши роботу комп'ютерних мереж під атаками та мотиваціями та поведінкою неавторизованого користувача Інтернету – зловмисного користувача або хакера [2].

Хакерська діяльність має дві основні сторони – технологічні та психологічні. Технологічні характеристики розкривають вид і спосіб злому хакерів, до яких можна віднести [3]:

1. Прослуховування: використання аналізаторів пакетів для спостереження за моделями трафіку;
2. Відстеження: неавторизоване завантаження даних, зловмисник прослуховує трафік між двома машинами в комп'ютерній мережі;
3. Втручання або підробка даних, коли зловмисник вносить неавторизовані зміни в дані або програмне забезпечення в системі;
4. Спуфінг: акт маскування повідомлення з невідомого джерела як повідомлення з відомого, надійного джерела;

5. Впровадження зловмисного коду: коли зловмисник використовує програмне забезпечення для перевірки вхідних даних, щоб ввести зловмисний код;

6. Використання недоліків у проектуванні, реалізації або роботі системи: уразливості системи — це недоліки в програмній системі або ж сама архітектура та специфікація;

7. Взлам паролів, кодів і ключів, відомий як атаки грубої сили. Для досягнення цілей проникнення можна застосувати два або більше методів у даній хакерській атаці.

Наприклад, хакери можуть застосувати таку послідовність атак: глушіння, спуфінг, впровадження шкідливого коду. Аналітики визначають хакерство як надмірний інтерес до комп'ютерних систем і мереж, електронних пристроїв і технологічних інновацій. Вторгнення в зони обмеженого доступу завжди було частиною перевірки здатності машини та її операторів [4, 5]. Існує два терміни, хакери та крекер (зламувачі), щоб розрізняти поведінку зловмисників на основі їхніх намірів діяти на інформаційних територіях з обмеженим доступом [6, 7]. Перший термін розкриває концепцію радісної насолоди та задоволення від елітної конкуренції, другий термін підкреслює кримінальну сутність наміру проникнення в комп'ютерну мережу.

Безпека є важливою проблемою для всіх мереж компанії та установи в даний час і все вторгнення намагаються таким чином, щоб отримати успішний доступ до мережа передачі даних цих компаній і веб-сервіси незважаючи на розробку багатьох способів забезпечити це проникнення або вторгнення в інфраструктуру мережі через Інтернет, за допомогою брандмауерів, шифрування тощо[8].

Зловмисне програмне забезпечення або зловмисний код мають намір завдати шкоди комп'ютеру системи без відома користувачів системи [9]. Зловмисне програмне забезпечення несвідомо встановлюється наївними

користувачами під час перегляду Інтернету [10]. Після встановлення шкідливі програми виконують ненавмисні дії, наприклад: а) викрадають ім’я користувача, пароль; б) встановлюють шпигунське програмне забезпечення для забезпечення віддаленого доступу до зловмисників; в) переповнюють спам-повідомлення систему; г) виконувати атаки на відмову в обслуговуванні, розподілені атака на відмову в обслуговуванні (DoS, DDoS, (distributed) denial-of-service) тощо. З появою метаморфічного шкідливого програмного забезпечення (яке використовує складні методи обфускації) детектори на основі сигнатур не можуть ідентифікувати нові варіанти шкідливого програмного забезпечення [11].

Метою зловмисної активності є одна або більше дій, таких як загрози ідентифікації, споживання системних ресурсів, надання несанкціонованого доступу до скомпрометованих систем [12]. Загальною його характеристикою є здатність до тиражування та розповсюдження. Зловмисне програмне забезпечення використовує файли, електронну пошту, макроси, Bluetooth або браузер як джерело зараження для свого поширення. Багато антивірусних продуктів використовують сигнатури для виявлення шкідливих програм. Підпис — це унікальний байтовий шаблон або рядок, здатний ідентифікувати код як шкідливий. Хоча цей метод успішно ідентифікує відомі шкідливі програми, але не може виявити невидимі зразки. Методи, засновані на сигнатурах, мають певні обмеження щодо виявлення:

a. неможливість виявити зашифрований код;
b. відсутність знань про семантику програми;
c. збільшення розміру сховища сигнатур;
d. неможливість виявити затуманене зловмисне програмне забезпечення.

Щоб обійти метод виявлення на основі шаблонів, розробники зловмисного програмного забезпечення використовують складні механізми обфускації для створення нових нащадків. Основою створення метаморфічного шкідливого ПЗ є збільшення варіативності структури коду в наступних поколіннях без зміни

функціональності програм. Евристичні методи виявлення також привернули багато досліджень. У цьому методі для ідентифікації невідомого шкідливого програмного забезпечення готується набір правил, що ефективно описує ознаки зловмисної активності. Набір правил відображає поведінку шкідливого коду. Недоліком евристичного методу є те, що він може повідомити про зловмисність зразків лише після того, як зловмисний код заразить систему[13].

Цілями для виконання зловмисної активності зазвичай є:

- Домашні користувачі: Домашні користувачі часто зберігають на комп'ютері конфіденційну інформацію, файли та документи, які є особистими цінностями, як-от проекти коледжу, фотографії та файли збереження відеоігор. Незважаючи на те, що ці речі є цінними для користувачів, домашні користувачі все ще навряд чи матимуть ефективну резервну стратегію для успішного відновлення після таких подій, як пожежа чи крадіжка, не кажучи вже про атаки крипто-вимагачів [14].

- Підприємства: бізнес-комп'ютери також частіше містять конфіденційні дані та документи критичної важливості, такі як бази даних клієнтів, бізнес-плани, пропозиції, звіти, вихідний код, форми та документи щодо дотримання податкового законодавства. Сучасні крипто-вимагачі можуть нумерувати всі доступні диски, такі як локальні файлообмінні сервери, і також шифрувати файли на них. Це означає, що лише одне зараження крипто-вимагачем може вплинути на кілька систем[15].

- Державні установи: державні установи, такі як навчальні заклади та навіть правоохоронні органи, не виключені з поля уваги цих кіберзлочинців, і в деяких випадках вони можуть бути спеціально націленими [16].

Методи виявлення зловмисного програмного забезпечення можна загалом класифікувати як статичні та динамічні. За допомогою статичного аналізу зловмисне програмне забезпечення виявляється шляхом перевірки коду або його структури без виконання зразків. Таким чином, статичний аналіз є швидким, але

може не виявити частини шкідливого коду під час виконання. За допомогою статичного аналізу сканер перевіряє рядки, імена файлів, підписи авторів, системну інформацію, контрольну суму тощо, які відрізняють шкідливе програмне забезпечення від безпечної програми. У динамічному аналізі зловмисне програмне забезпечення виконується в контрольованому середовищі. Сканери, що використовують цей метод, відстежують функції/системні виклики, стан регістрів процесора, прапори, параметри API, щоб ідентифікувати програму як шкідливу [17].

Хоча динамічний аналіз, як правило, повільний і не може використовуватися як самостійний механізм для виявлення зловмисного програмного забезпечення, саме тому сканер намагається відстежити шляхи виконання підозрюваного зразка [18]. Зараження систем є основним ризиком, пов’язаним з динамічним аналізом. Щоб уникнути цього, сканери зловмисного програмного забезпечення використовують методи віртуалізації або емуляції [19]. Це знижує ефективність, оскільки збільшується час виконання. Динамічний аналіз може бути невдалим, якщо зловмисне програмне забезпечення містить перевірки проти віртуальної машини та емуляції. Методи інтелектуального аналізу даних також набувають популярності у виявленні шкідливих програм. У цьому методі алгоритм класифікації використовується для моделювання зловмисного програмного забезпечення та доброякісної поведінки/структури. Класифікатор піддається різноманітним шкідливим і доброякісним шаблонам для класифікації зразків (зловмисне програмне забезпечення або доброякісне). Останнім часом методи активного навчання [20] також використовуються в області виявлення шкідливих програм. Ця методика в основному використовується для оцінки ефективності моделей, навчених раніше, порівняно з новими тестовими зразками. Необхідно періодично оновлювати навчені моделі, щоб справлятися з невидимим шкідливим програмним забезпеченням.

Експерименти проводяться на портативних виконуваних (PE) файлах Win32 як зловмисних, так і безпечних зразків.

На жаль, важко зрозуміти масштаби потенційних загроз викликаних зловмисною активністю в мережі через децентралізацію Інтернету. Крім того сучасні технології невпинно розвиваються, що приводить до того, що питання виявлення вторгнень стає дуже актуальним.

1.2 Розподілені системи виявлення зловмисного програмного забезпечення

Розподілені системи виявлення зловмисного програмного або DIDS загалом складаються з кількох систем виявлення вторгнень у великій мережі, усі з яких обмінюються даними або з центральним сервером, який забезпечує розширений моніторинг[21]. У розподіленому середовищі DIDS реалізуються за допомогою кооперативних інтелектуальних агентів, розподілених по мережі [22].

Сфера розподіленого середовища значно розвивалася за останні два десятиліття, надаючи послуги програмного та апаратного забезпечення різним користувачам, що змінило обчислювальне середовище спільного використання даних, спільного циклу та інших послуг з точки зору розподілених ресурсів [23]. Вторгнення визначається як кожна група дій, спрямованих на загрозу конфіденційності, цілісності або доступності ресурсів [24]. В інформаційній системі вторгнення означає будь-яку діяльність, яка порушує політику безпеки системи. У той час як виявлення вторгнень являє собою процес, який використовується для ідентифікації вторгнень, виявлення вторгнень базується на поведінці зловмисника, яка, зрозуміло, буде відрізнятися від законної, отже, значення інформаційних систем як комплексних активів для організацій. Незважаючи на те, що підсистеми комплексного виявлення вторгнень зазвичай не є додатками, їх було включено як елементи операційних систем [25]. Майже всі системи виявлення вторгнень намагаються виявити ймовірні вторгнення, а

потім сповістити системного адміністратора. Технології автоматизованого реагування на вторгнення знаходяться лише на початковій стадії розробки. Оригінальні системи виявлення вторгнень передбачали, що система єдиного автономного процесора та методів виявлення складалася з постфактум обробки записів аудиту, тоді як поточні системи складаються з різних вузлів, які виконують кілька операційних систем, які пов’язані між собою, утворюючи єдину розподілену систему[26,27].

Крім того, вторгнення можуть бути здійснені декількома зловмисниками, що може значно збільшити складність без фундаментальних проблем [28]. Зазвичай системи виявлення вторгнень (IDS) розгортаються з іншими механізмами превентивної безпеки, такими як автентифікація та авторизація. IDS діють як друга лінія захисту для захисту інформаційних систем. Однак є багато причин, які можуть зробити виявлення вторгнень необхідною частиною системи захисту. Наприклад, численні класичні додатки та традиційні системи були розроблені та розширені без урахування безпеки на етапах проектування та впровадження, і системи, і додатки були розроблені для роботи в різноманітних середовищах, тому вони можуть бути вразливими під час розгортання в існуючому середовищі. Система може бути безпечною в ізольованому стані, однак вона стає вразливою під час підключення до Інтернету. Таким чином, функція виявлення вторгнень може ідентифікувати атаки проти цих систем і, відповідно, дозволити реагувати на них. Крім того, відсутність інформаційної безпеки та методів розробки програмного забезпечення може призвести до помилок або недоліків, які можуть бути використані зловмисником для атаки на системи. Отже, механізми запобігання, такі як брандмауери, може бути не таким ефективним, як очікувалося[29].

Оскільки зловмисники та методи атак стають дедалі складнішими, потреба в системі DIDS у великих корпоративних і військових мережах різко зростає [30]. Зі збільшенням складності цих атак аналітики залишаються відкритими для

проблем комунікаційних збоїв, коли один аналітик бачить одну атаку на свій сегмент і відкидає її як ніщо. У той час як кілька інших сегментів отримують ті самі атаки скоординовано, аналітики можуть нехтувати серйозністю атаки. Однак, коли всі дані атаки розглядаються разом, може виникнути кардинально інша перспектива атаки. Система DIDS дає аналітику швидший, простіший і ефективніший метод виявлення скоординованих атак у кількох сегментах мережі та відстеження діяльності зловмисників. Зрештою, система також економить кошти підприємства або користувача, у мережах якої вона розгорнута, зменшуючи кількість необхідних інцидентів, а також кількість часу, необхідного для збору журналів із різних налаштувань систем IDS у великій корпоративній мережі [31]. Завдяки тому, що всі ці записи про атаки зберігаються в одному місці, це дає аналітику набагато більшу гнучкість у виявленні моделей атак та інших проблем з атаками, які в іншому випадку могли б залишитися непоміченими.

Дослідження DIDS у літературі почалися в 1990-х роках і тривали широко до 2009 року, але потім зменшилися [32]. Після цих років стало видно, що дослідження IDS здебільшого зосереджені на бездротових сенсорних мережах і хмарі. Розподілена система виявлення вторгнень використовувала комбінацію моніторів хоста та локальної мережі для спостереження за діяльністю системи та мережі. Централізоване ядро отримувало інформацію від системи для виявлення вторгнень. Номенклатура CIDF, згадана вище, включає агентів розвідки для збору даних, агентів аналізу та агентів прийняття рішень [33]. Проект «Комп’ютерна імунологія» в Університеті Нью-Мексико [34] досліджував структуру IDS, засновану на ідеях, отриманих шляхом дослідження імунної системи тварин. Одна частина проекту розвинула самосвідомість комп’ютерних програм, пов’язаних із безпекою, шляхом спостереження за звичайними наборами системних викликів, які виконуються програмами. Це самосвідомість можна використовувати для виявлення вторгнень шляхом того, що програма

виконує незвичайний набір системних викликів. Проект JAMP у Колумбійському університеті [35] використовує інтелектуальні розподілені Java-агенти та інтелектуальний аналіз даних для вивчення моделей шахрайства та нав'язливої поведінки, які можуть використовуватися між організаціями.

Якщо згадати переваги розподілених систем, то можна вказати, що вони полягають у тому, як система пропонує аналітику інцидентів. Це надає їй привілеї перед іншими одно-режимними системами IDS [36]. Варто виділити можливість виявлення шаблонів атак у всій корпоративній мережі з географічним розташуванням, розділеним на сегменти часовими поясами або навіть континентами. Це може дозволити завчасно виявити добре сплановану та скоординовану атаку на відповідну організацію, що дасть змогу спеціалістам із безпеки переконатися, що цільові системи захищені, а IP-адресам-порушникам заборонено будь-який доступ. Ще однією перевіреною перевагою є можливість раннього виявлення Інтернет-хробака, який може пробратися через мережу [37]. Потім ця інформація може бути використана для ідентифікації та очищення систем, які були заражені хробаком, і запобігання подальшому поширенню хробака в мережі, таким чином зменшуючи будь-які фінансові втрати, які могли б бути понесені в іншому випадку.

Друга велика перевага полягає в тому, що тепер одна група аналізу може робити те, що раніше вимагало кількох груп аналізу інцидентів через фізичну відстань [38]. Це позбавляє від необхідності платити за окремі групи аналізу інцидентів для кожного окремого географічного розташування джерел мережі. Іншою проблемою, яку він вирішує, є атаки мережі корпорацій з боку розлючених, засмучених або просто нудьгуючих працівників. Такий тип загрози називають інсайдерські атаки [39].

Розподілені системи виявлення вторгнень є наступним логічним рівнем для систем IDS. Їх можна налаштувати з уже існуючими архітектурами та системами IDS, що робить систему ще більш економнішою. Слід також зазначити, що наразі

існує кілька систем, які відповідають принципам dIDS. Однією з таких служб є служба ARIS Predictor від SecurityFocus, яку слід відзначити завдяки широкомасштабній демонстрації використання DIDS у звітах про атаки провайдерам Інтернет-послуг, власникам серверів, а також їх доведеній здатності ідентифікувати нових черв'яків і атаки, таким чином доповнюючи концепцію безпечного інтернету.

Система DIDS надають аналітику набагато швидше, простіший та ефективніше у кількох сегментах мережі для відстеження дій зловмисників [40]. Зрештою, система також економить кошти підприємств або корпорацій, у мережах якої вона розгорнута, зменшуючи кількість необхідних аналітиків інцидентів, а також кількість часу, необхідного для збору журналів із різних налаштувань систем IDS у великій корпоративній мережі [41]. Завдяки тому, що всі ці записи про атаки зберігаються в одному місці, система має набагато більшу гнучкість у виявленні шаблонів атак та інших проблем з атаками, які в іншому випадку могли б залишитися непоміченими.

Оскільки зловмисні методи атак стають дедалі складнішими, потреба в системі DIDS у великих корпоративних і військових мережах різко зростає. Зі збільшенням складності цих атак системи залишаються відкритими для проблем комунікаційних збоїв, коли одна система бачить одну атаку на свій сегмент і просто відкидає її. Паралельно як кілька інших сегментів отримують ті самі атаки скоординовано, їхні системи можуть не відкидати серйозність атаки.

1.3Засоби та методи штучного інтелекту для задач прогнозування

Штучний інтелект (ШІ) часто називають галуззю науки, яка допомагає машинам знаходити рішення складних проблем більш схожим на людину способом [42, 43]. Його зазвичай асоціюють з інформатикою, але він має багато важливих зв'язків з іншими галузями, такими як математика, психологія,

пізнання, біологія та філософія. Успіх штучного інтелекту пояснюється тим, що його технології поширилися в повсякденному житті. Нейронні мережі, нечіткі засоби керування, дерева рішень і системи на основі правил уже є в наших мобільних телефонах, пральних машинах і бізнес-додатках [44].

Прогнозування Інтернет-трафіку також є дуже важливим для таких завдань, як розподіл ресурсів, планування мережі та виявлення мережевих аномалій, спричинених атаками [45, 46]. Точну модель прогнозування можна використовувати для виявлення атак на безпеку в комп'ютерних мережах шляхом порівняння прогнозованого трафіку з фактичним. [47] Крім того, прогнозування майбутнього трафіку в комп'ютерній мережі на основі поточного трафіку дозволяє мережевому менеджеру вживати заходів до нападів, перевантажень, розривів з'єднання або простоїв. Подібні прогнози можна зробити шляхом моделювання трафіку вхідних і вихідних даних мережі як часових рядів. В даний час існує кілька досліджень у цій галузі [48, 49].

Зі збільшенням використання Інтернету важливість класифікації даних в Інтернеті та підвищення ефективності використання Інтернету зросла, особливо для менеджерів мереж, які керують корпоративними мережами. Останнім часом набули поширення дослідження з класифікації інтернет-трафіку. Для цих цілей дуже важливими стали методи машинного або глибокого навчання та класифікація мережевого трафіку, і їх вплив зростає з кожним днем. Для класифікації мережевого трафіку існують різні підходи, засновані на методах на основі портів, на основі навантаження, на основі сервера, на основі IP або потоку [50]. Існує багато успішних досліджень класифікації трафіку на основі контрольованого потоку [51], [52], [53], [54], [55], [56], [57]. Дослідження Roughan є одним із перших досліджень класифікації мережевого трафіку з використанням методів навчання під керівництвом [51]. У їх дослідженні використовувалися k-NN і метод лінійного дискримінантного аналізу, щоб узгодити мережевий трафік відповідно до вимог якості послуг (QoS). Не завжди

можливо зібрати двонаправлені потоки, але можна вивести компоненти вектора ознак для даного класу додатків або передбачити стару статистику [58], [59]. k-NN і метод лінійного дискримінантного аналізу використовувалися для узгодження мережевого трафіку відповідно до вимог якості послуг (QoS). Не завжди можливо зібрати двонаправлені потоки, але можна вивести компоненти вектора ознак для даного класу додатків або передбачити стару статистику. k-NN і метод лінійного дискретного аналізу використовувалися для узгодження мережевого трафіку відповідно до вимог якості послуг (QoS).

Використання минулих спостережень для прогнозування майбутнього мережевого трафіку є важливим кроком для розуміння та керування мережею комп’ютера [60]. Прогноз трафіку комп’ютерної мережі має вирішальне значення для мережевих провайдерів і комп'ютерної мережі і управління в цілому. Він представляє значний інтерес у кількох областях, таких як адаптивні програми, контроль перевантаження, контроль доступу та розподіл пропускної здатності. Є багато досліджень, які зосереджуються на адаптивних і динамічних додатках. Зазвичай вони представляють деякі алгоритми, які використовують навантаження трафіку для динамічної адаптації пропускної здатності певного компонента мережі [61] і покращують якість обслуговування (QoS) [62]. Найбільш конкурентоспроможною моделлю, яка долає класичні методи регресії, такі як авторегресійне інтегроване ковзне середнє (ARIMA) [63]. Таким чином, є роботи, які поєднують ці два фактори, створюючи передбачувану нейронну мережу для динамічного розподілу пропускної здатності в реальному.

Мережевий трафік – це часовий ряд, який є послідовністю даних, які регулярно вимірюються через однакові проміжки часу [64, 65]. Для мережевого трафіку ці послідовні дані є бітами, що передаються деяким мережевим пристроєм у певний період часу. Часовий ряд може бути стохастичним процесом або детермінованим. Щоб передбачити часовий ряд, необхідно використовувати

математичні моделі, які справді представляють статистичну характеристику вибіркового трафіку.

ANM або штучні нейронні мережі — це прості структури обробки, які розділені на міцно пов’язані одиниці, які називаються штучними нейронами (вузлами)[66]. Нейрони організовані в шари, один шар містить кілька нейронів, і будь-яка одна нейронна мережа може мати один або більше шарів, які визначаються топологією мережі та відрізняються в різних мережевих моделях. Нейрони здатні працювати паралельно для обробки даних, зберігання експериментальних знань і використання цих знань щоб отримати нові дані. Кожен нейрон має синаптичну вагу, яка відповідає за зберігання отриманих знань. Знання мережі набуваються через процеси навчання (алгоритм навчання або мережеве навчання) [67]. У процесі навчання нейронна мережа буде навчена розпізнавати та відрізняти дані від кінцевого набору. Після навчання ШНМ готова розпізнавати закономірності, наприклад, у часовому ряді. Під час процесу навчання синаптичні ваги змінюються впорядкованим чином, поки вони не досягнуть бажаного результату. Нейронна мережа пропонує ті самі функції, що й нейрони в мозку людини, для вирішення складних проблем, таких як нелінійність, високого паралелізму, надійності, відмовостійкості і шумостійкісті, адаптивності, навчання та узагальнення [67,68].

DL або глибоке навчання відноситься до методу машинного навчання, який базується на моделі нейронної мережі з кількома рівнями представлення даних. Ієрархічні рівні представлення організовані абстракціями, ознаками або поняттями. Вищі рівні визначаються нижчими рівнями, де представлення низьких рівнів може визначати кілька різних характеристик високих рівнів, це робить представлення даних більш абстрактним та нелінійним для вищих рівнів [69, 70]. Ці ієрархічні рівні представлені рівнями ШНМ.

Після визначення різних етапів процесу навчання та структури ШНМ, а також до розробки процедури оптимізації, важливо оцінити якості прогнозування ШНМ.

Таблиця 1. Особливості ШІ для задач прогнозування

№	Особливість	Опис
1	адаптивність	Повторення функцій роботи людського мозку
2	самоорганізація	Накопичення досвіду виражається у зміні структури
3	паралельна робота	Паралельна робота усіх процесів
4	відмовостійкість	Здатність працювати на основі неповних даних
5	швидкість	В порівнянні з іншими системами працює швидше
6	час	Адаптований час виконання
7	класифікація	Використовується для класифікації даних
8	характер даних	Використовується коли точно відомі дані

Результати моделі ШНМ порівнюються з даними гідродинамічного моделювання [76].

Особливості використання ШІ для задач прогнозування (таблиця 1):

1. Адаптивне навчання: ШНМ повторює людський мозок у тому, як він вчиться виконувати завдання під час навчання. Звичайна програма не може адаптуватися до інших типів вхідних даних [77];

2. Самоорганізація: ANN може створити власну організацію під час навчання. Накопичення даних (досвіду) виражається у зміні структури [78];

3. Паралельна робота: ШНМ працює паралельно, як людський мозок. Це не схоже на комп’ютерну програму, яка працює серійно [79];

4. Відмовостійкість. Однією з найцікавіших властивостей нейронних мереж є їх здатність працювати навіть на основі неповних, зашумлених і нечітких

даних. Звичайна програма не може обробляти неповні, незрозумілі дані і зупиниться працювати, коли зустрічає найменші неправильні дані [80];

5. Швидкість. У порівнянні з людським мозком ШНМ досить швидка, оскільки час обробки людським мозком повільніший;

6. Час. Порівняно зі звичайною програмою метод, за допомогою якого ШНМ обчислює вихід, незрозумілий. Витрачений час постійно змінюється з різними наборами вхідних даних, хоча вони схожі;

7. Класифікація. ШНМ можна використовувати для класифікації даних, розпізнавання образів і в програмах, де дані нечіткі;

8. Характер. ШНМ можна використовувати, коли точно відомий характер вхідних і вихідних даних і чітко відомо, що потрібно зробити.

Штучний інтелект дозволяє додавати значну складність моделі прогнозування. Основна складність використання ШІ відноситься до етапу підготовки. Звичайні алгоритми, такі як зворотне поширення, не працюють добре, якщо нейронна мережа має більше трьох прихованих рівнів [71]. Крім того, ці звичайні алгоритми не оптимізують використання більшої кількості рівнів і не розрізняють характеристики даних ієрархічно, тобто нейронна мережа з багатьма шарами не має кращого результату, ніж нейронна мережа з кількома шарами, наприклад, неглибока нейронна мережа з двома або трьома шарами [72, 73].

Одним з найбільш актуальних аспектів ШІ є його здатність узагальнювати, тобто передбачати випадки, які такими не є входять до навчального набору [74]. Одна з проблем, які виникають під час навчання нейронної мережі називається перепідгонкою. Помилка на навчальному наборі доведена до дуже малого значення, але коли нові дані представлені в мережу помилка стала більшою. Мережа запам'ятала тренувальні приклади, але не навчилася узагальнювати нові ситуації. Одним із методів покращення узагальнення мережі є використання мережі, яка є достатньо великою, щоб забезпечити адекватну підгонку.

Чим більшу мережу ви використовуєте, тим складніші функції мережа може створити. Існує два інші методи покращення узагальнення, які реалізовані в програмному забезпеченні Mat Lab Neural Network Toolbox: регулярізація та рання зупинка. Типовою функцією продуктивності [75], яка використовується для навчання нейронних мереж прямого зв'язку, є середня сума квадратів помилок мережі (1.1),

$$mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{real}(i) - x_{pred}(i))^2 \quad (1.1)$$

Можна покращити узагальнення, якщо змінити функцію продуктивності, додавши член, який складається із середнього значення суми квадратів мережевих ваг і зміщень (2),

$$mse_{reg} = \lambda mse + (1 - \lambda)msw \quad (1.2)$$

де λ – коефіцієнт ефективності, і (3):

$$msw = \frac{1}{N}\sum_{j=1}^{N} w_j^2 \quad (1.3)$$

Використання цієї функції продуктивності призводить до того, що мережа матиме менші ваги та зміщення, і це змушує відповідь мережі бути плавнішою та використовувати меншу ймовірність перевиконання.

1.4 Постановка задачі дослідження

Головна задача магістерської роботи полягає в проектуванні самоорганізованої розподіленої системи прогнозування зловмисної активності шляхом виконання наступних підзадач:

- проведення аналізу систем прогнозування зловмисної активності;
- дослідження особливостей функціонування зловмисного програмного забезпечення в комп'ютерній мережі;
- спроектувати систему виявлення вторгнень яка буде відповідати критеріям розподіленості та самоорганізованості;

- розробити метод виявлення шкідливого трафіку на основі впровадження нейронних мереж та глибокого навчання;
- реалізувати спроектовану систему шляхом розробки програмного забезпечення;
- провести дослідження ефективності побудованої системи шляхом перевірки її на тестових даних.

Висновки до першого розділу

З постійним зростанням потенційних загроз і атак є досить складним завданням точно і своєчасно виявити зловмисну або потенційно небезпечну активність у комп'ютерних мережах. Сучасні технологій пропонують величезну кількість принципів, методів, систем для виявлення такої активності. Однак на своєму шляху вони зустрічаються з критичними проблемами через активні постійно зростаючі нові загрози, які теперешні системи не розуміють. Сьогодення показує, що для попередження зловмисної активності, використовуються системи виявлення вторгнень. Такі системи зазвича працють з попередньо встановленими методами, засновані на конкретних правилах (сигнатурах) або на методах прогнозованого моделювання. Тому, для вирішення цих проблем потрібно побудувати систему. Найчастіше такі методи використовують відомі атаки як базову лінію для порівняння поведінки трафіку, але такі системи не враховують нові шаблони. Для вирішення цієї проблеми потрібно побудувати розподілену систему яка зможе реагувати на ці невідомі шаблони. Одним з найефективніших проявів даної системи є використання систем виявлення вторгнень. Саме система IDS на основі методів штучного інтелекту забезпечити ефективну роботу системи по виявленню зловмисного (шкідливого) трафіку у мережі.

2 ЗЛОВМИСНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТА ЙОГО ФОРМАЛІЗАЦІЯ

2.1 Характеристика зловмисного програмного забезпечення

Шкідливе програмне забезпечення, або скорочено ЗПЗ, — це програмне забезпечення, розроблене з підлим наміром завдати шкоди користувачеві комп’ютера. Існує багато типів зловмисного програмного забезпечення, залежно від того, як вони поширюються та характеру шкоди, яку вони заподіюють. Деякі приклади зловмисного програмного забезпечення включають віруси, хробаки, траяни, шпигунське ПЗ, клавіатурні шпигуни, ботнети, руткіти, програмне забезпечення-вимагач (ransomware), відлякувальне (scareware) ПЗ та автозавантажувачі. На сьогоднішній день виявлено більше мільйона різних вірусів та інших шкідливих програм. Деякі з них завдали значних збитків окремим особам і організаціям, іноді на мільярди доларів США. Деякі відомі віруси в хронологічному порядку включають черв’яка Morris у 1988 році, вірус Melissa у 1999 році, вірус ILOVEYOU у 2000 році, вірус Anna Kournikova у 2001 році, черв’як Code Red у 2001 році, вірус Slammer у 2003 році, черв’як Mydoom у 2004 р., хробаки Sasser і Netsky у 2004 р., хробак Storm у 2007 р., зловмисне програмне забезпечення Mirai у 2016 р. та WannaCryransomware у 2017 р. Шкідливе програмне забезпечення, яке завдало найбільшої шкоди на сьогоднішній день, — це хробаки Sasser і Netsky, збиток яких оцінюється в 31 мільярд доларів США. Іноді навіть уряди схильні використовувати зловмисне програмне забезпечення для шпигунства та інших політичних мотивів. Запобігти впливу зловмисного програмного забезпечення можна використовуючи відповідне ПЗ безпеки, таке як брандмауери, антивірусне програмне забезпечення та антишпигунське програмне забезпечення. Крім того, дослідники використовували кримінологічні теорії, зокрема теорії самоконтролю та

повсякденної діяльності, щоб визначити фактори, які можуть підвищити ризик зараження шкідливим програмним забезпеченням. Існуючі дані свідчать про те, що безвідповідальне використання Інтернету, як-от невикористання програмного забезпечення яке призначене для комп'ютерної безпеки або натискання на сумнівні веб-сайти, також може призвести до зараження зловмисним програмним забезпеченням. Відповідно, щоб ефективно боротися зі зловмисним програмним забезпеченням, технічні аспекти проблеми, а також антропогенна сторона проблеми повинні бути одночасно розглянуті.

Шкідливі програми традиційно вважаються чисто технічними загрозами, які використовують технічні вразливості системи ІКТ для здійснення атаки. У середовищі Інтернету та хмарних обчислень, щоб обманути користувачів хакери використовують соціальні мережі, нові технології атак і більш складні шкідливі програми. Головною метою атаки є отримання певної фінансової та іншої вигоди. Організовані злочинні угруповання, терористичні організації та агресивні держави є зловмисниками, а не хакерами-одинаками.

Розробники ЗПЗ стають розумнішими з точки зору своєї здатності створювати зловмисне програмне забезпечення, яке не буде виявлено програмним забезпеченням для захисту від зловмисного програмного забезпечення, і розробникам захисного ПЗ потрібно постійно залишатися інноваційним, щоб боротися з розумнішим ЗПЗ.

ЗПЗ - комп'ютерне програмне забезпечення, створене з наміром завдати шкоди комп'ютеру, комп'ютерній мережі, серверу, мобільному пристрою, будь-якому взаємопов'язаному пристрою/системі (медичні пристрої, автомобілі, літаки, електрична мережа тощо), або користувачам комп'ютерів. Після встановлення на комп'ютері цілі в оманливий спосіб ці програми можуть серйозно поставити під загрозу систему безпеки комп'ютера, а також конфіденційність власника. ЗПЗ може завдати шкоди багатьма способами, наприклад: отримання несанкціонованого доступу до комп'ютерних систем і

мережевих ресурсів, порушення роботи комп'ютера та/або збір особистої інформації власника системи без його чи її відома чи згоди. ЗПЗ також створюється та використовується для отримання фінансової вигоди, вимагання та вчинення різних видів злочинів. Наприклад, у період з вересня 2013 року по травень 2014 року комп'ютерний вірус, відомий як CryptoLocker, був представлений і поширювався в Інтернеті через вкладення електронної пошти. Багато хто вважає, що ця кібератака була спрямована на комп'ютери, які працюють під керуванням Microsoft Windows, і після активації вона зашифрувала певні типи файлів, що зберігаються на ураженому комп'ютері (тобто вона закодувала інформацію чи дані в цих файлах, щоб отримали до них доступ лише авторизовані особи).

Шкідливе програмне забезпечення зазвичай має форму виконуваних кодів або сценаріїв. Зокрема, основною метою зловмисного програмного забезпечення є використання наявних вразливостей у веб-службах, браузерах, операційних системах і комп'ютерних програмах для особистої, фінансової чи політичної вигоди. Хоча комп'ютерні системи та веб-браузери мають різні вразливості та існують різні способи їх використання. Два загальні підходи, пов'язані зі зловмисним програмним забезпеченням, включають маніпулювання комп'ютером користувача для виконання зловмисних дій (наприклад, поширення комп'ютерного вірусу на інші комп'ютерні системи через повідомлення електронної пошти) або використання комп'ютера користувача як зброю для атаки на інші системи (наприклад, розподілені атаки на відмову в обслуговуванні DDoS). Таким чином, ЗПЗ є основною платформою для онлайн-злочинців, а також основною загрозою безпеці, з якою сьогодні стикається Інтернет. Загалом, більшість користувачів комп'ютерів навіть не підозрюють, що на їхніх комп'ютерах встановлено ЗПЗ. ЗПЗ зазвичай встановлюється на комп'ютерну систему випадково. Подібним чином користувач комп'ютера може завантажити безкоштовне програмне забезпечення з Інтернету та в процесі ненавмисно

встановити шкідливе програмне забезпечення у своїй системі. Зловмисне програмне забезпечення також може заражати комп'ютерну систему, коли користувач нехтує оновленням свого програмного забезпечення безпеки (наприклад, антивірусного програмного забезпечення, антишпигунського програмного забезпечення та брандмауерів). Слід зазначити, що зловмисне програмне забезпечення іноді додається до законного програмного забезпечення та завантажується в комп'ютерну систему без відома чи згоди користувача. Наприклад, у 2005 році Sony Corporation зіткнулася з кількома колективними позовами після того, як було виявлено її оманливі заходи захисту від копіювання. Зокрема, щоб запобігти нелегальному копіюванню її музичних компакт-дисків своїми клієнтами, Sony вбудувала у свої компакт-диски програмне забезпечення, яке приховувало себе після встановлення на комп'ютері покупця.

2.2 Класифікація проявів зловмисної активності

Зловмисна активність у сучасних комп'ютерних існує у різних проявах і для правильного розуміння способів попередження зловмисної активності потрібно розглянути їх форми:

1. Комп'ютерні віруси, мабуть, є найдавнішим відомим типом шкідливого програмного забезпечення. Aviruse — це самовідтворювана програма, створена для поширення та зараження нових хостів, як біологічний вірус.

Перші екземпляри вірусів були досить нешкідливими, але перший по-справжньому небезпечний вірус, відомий як Festering Hate virus, був випущений у 1988 році та був націлений на операційні системи Apple. Цей вірус не тільки заразив кожен файл на жорсткому диску, дискетах і накопичувачах пам'яті, він також знищив усі файли. Сучасні нащадки ранніх вірусів розвиваються та поширюються скрізь. Те, що колись було просто кібер-вандалізмом і лиходійством, швидко перетворилося на кіберзлочинність. Сьогодні віруси

створюються зі зловмисними та злочинними намірами. Після того, як вірус заразив комп'ютерну систему, він потім поширюється на інші системи різними способами, наприклад через вкладення електронної пошти, надіслані користувачам в адресну книгу зараженого комп'ютера.

Віруси також можуть використовувати ідентифікаційні дані зараженої системи або хоста, щоб створити електронний лист, який нібито надходить із надійного джерела, і надіслати його новим одержувачам. Існує два типи вірусів: вірус завантажувального сектора та програмний вірус. Віруси завантажувального сектора призначені для зараження сектора операційної системи, який завантажується з жорсткого диска комп'ютера (тобто, частини завантажувального сектора дисків комп'ютера), тоді як програмні віруси розроблені з можливістю приховування та приєднання до виконуваних файлів. Коли виконувані коди запускаються за допомогою взаємодії користувача, вірус виконує зловмисну функцію, для якої він був розроблений (наприклад, видалення файлів або зараження інших файлів). Віруси також можуть бути записані для виконання своїх функцій у певну дату або після певної події.

2. Worms або комп'ютерний хробак. Терміни вірус і термін хробак, як правило, використовуються як синоніми, хоча між двома типами програм є чіткі відмінності. На рисунку 2.1 показано життєвий цикл комп'ютерного хробака. Зараження відбувається з допомогою впровадження у систему хробака через електронну пошту, різні шкідливі веб-сайти або фізичні носії (наприклад через флеш-пам'ять). На етапі зараження відбувається впровадження коду в систему. Етап затримки відповідає очікуванню системою умов зловмисника. Далі етап затримки поділяється на поширення/самовідновлення та зловмисну активність.

Хробаки відрізняються від вірусів тим, що вони не потребують жодних дій від користувача комп'ютера, щоб проникнути в інші комп'ютерні системи. Іншими словами, хробаки можуть поширюватися з одного комп'ютера на інший, не приєднуючись файлів системи. Хробаки також відрізняються від вірусів тим,

що після проникнення в комп’ютерні мережі чи системи вони здатні виживати самі по собі та розмножуватися в заражених мережах чи системах. Враховуючи наведені вище відмінності, хробаки, як правило, становлять більшу загрозу, ніж віруси.

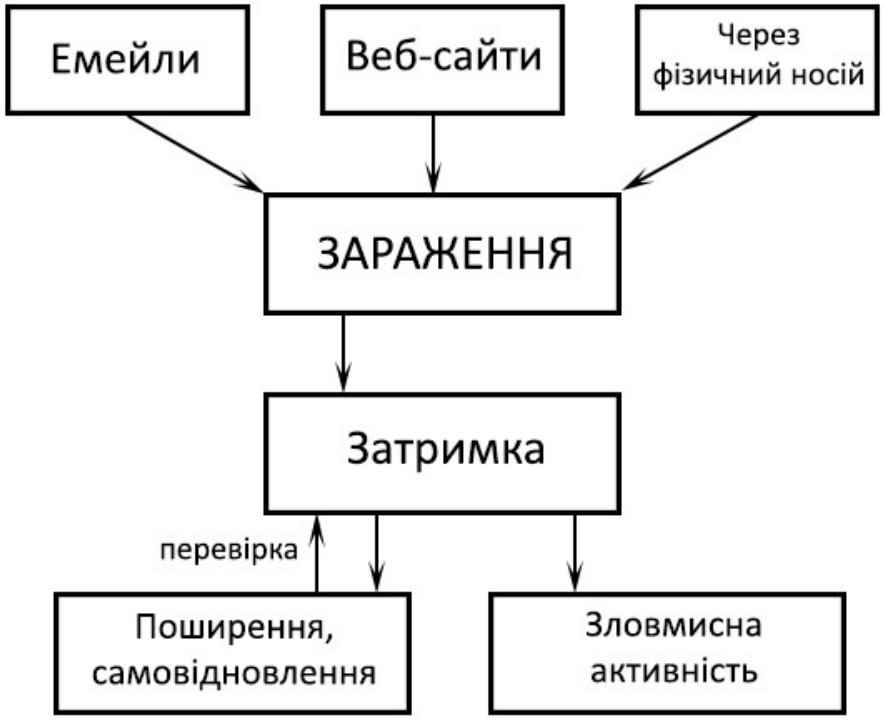


Рисунок 2.1 – Життєвий цикл комп’ютерного черв’яка в системі

3. Трояни або троянські коні. Подібно до вірусів і хробаків, троянські коні зазвичай мають приховану деструктивну функцію, яка активується під час виконання програми.

На рисунку 2.2 показано структуру загроз троянського коня у системі. Загроза рівня пристрою визначається як пошкодження пристрою, спричинене трояном. Загроза системного рівня проявляється як втручання в мережу та систему. Загрозу рівня даних загалом можна розуміти як ризик атак на дані,

особисті дані та конфіденційну інформацію. Загрози рівня додатків наголошують на загрозах додатків, загрозах керування та загрозах безпеці Інтернету речей.

Цей тип зловмисного програмного забезпечення отримав свою назву від давньогрецької історії про оманливого дерев’яного коня, який призвів до падіння міста Троя. За легендою, греки подарували троянцям дерев’яного коня, але всередині видовбаного коня були заховані грецькі воїни. Після того, як троянці прийняли подарунок і затягнули дерев'яного коня в стіни свого міста, сховані грецькі воїни виповзли звідти і напали на троянців, поки вони спали. Подібно до дерев’яного коня, який використовувався під час Троянської війни, зловмисне програмне забезпечення троянського коня маскується під корисну програму чи подарунок (тобто безкоштовне програмне забезпечення), і після того, як авторизований користувач встановить його на свій комп’ютер, зловмисне програмне забезпечення запускає руйнівний код, працює у фоновому режимі та виконує шкідливі завдання, на виконання яких вони були запрограмовані. Подібно до хробаків, троянські коні є автономними програмами, і вони не намагаються впровадити себе в інші файли чи іншим чином розповсюджувати себе. На відміну від хробаків, троян коні, як правило, поширюються через певну форму соціальної інженерії або через психологічне маніпулювання людьми для виконання певних дій. Зокрема, зловмисне програмне забезпечення троянського коня розроблено, щоб спонукати користувачів комп’ютерів запровадити його у свої комп’ютерні системи, видаючи себе за корисну та безкоштовну програму, яка користувачі хотіли б запустити (тобто як подарунок). Коли троянський кінь потрапляє в комп’ютерну систему, він часто діє як бекдор, який дозволяє оператору зловмисного програмного забезпечення отримати доступ і взяти під контроль уражений комп’ютер.

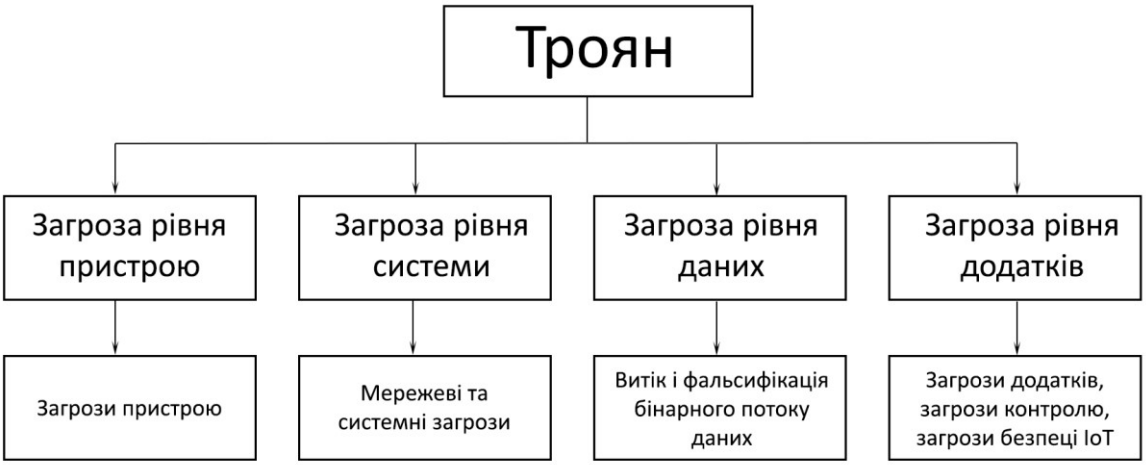


Рисунок 2.2 – Структура загроз троянів

4. Шпигунське програмне забезпечення — це тип шкідливого троянського програмного забезпечення, який використовується для збору та компіляції інформації про користувача комп’ютера чи групи користувачів без їхнього відома. Шпигунське програмне забезпечення також використовується для відстеження історії веб-перегляду користувачів комп’ютерів, а також тип «рекламного програмного забезпечення», яке відображає небажані спливаючі рекламні банери, спрямовані конкретно на користувача комп’ютера на основі його чи її історії веб-перегляду та файлів cookie. Дані, зібрані шпигунським програмним забезпеченням зазвичай містить особисту інформацію (наприклад, номер соціального страхування, номер водійського посвідчення), фінансову інформацію, облікові дані для входу та адреси Інтернет-протоколу (IP). Часто дані, зібрані шпигунським програмним забезпеченням, призначені для отримання фінансової вигоди або в рекламних цілях. Як правило, користувач комп’ютера заносить шпигунське програмне забезпечення у свою комп’ютерну систему, коли він або вона встановлює безкоштовне програмне забезпечення з Інтернет мережі. Користувач комп’ютера також може випадково занести

шпигунське програмне забезпечення у свою чи її комп’ютерну систему, коли він або вона відвідує заражені веб-сайти.

5. Кейлоггер. Клавіатурна шпилька — ще один тип шкідливого програмного забезпечення троянського коня. Подібно до шпигунського програмного забезпечення, кейлоггер використовується для збору особистої інформації або даних від користувачів комп’ютера без їх відома. Цей тип зловмисного програмного забезпечення реєструє або зберігає натискання клавіш, включаючи рухи миші, що створюється користувачем комп’ютера у файлі, а ця інформація згодом використовується для отримання особистої інформації, такої як номер соціального страхування, пароль і номер банківської картки.

6. Ботнети або мережі ботів. Ботнет — це група пристроїв, підключених до Інтернету, наприклад група комп’ютерів, на кожному пристрої з якими працює один або кілька веб-роботів. Веб-робот, або просто бот, — це програма, яка виконує автоматизовані завдання через Інтернет. Відмінною рисою бота є його здатність виконувати прості, повторювані завдання набагато швидше, ніж людина. Боти використовуються для різних цілей, найпопулярнішою з яких є веб-павук, також відомий як веб-сканування, механізм, який дозволяє пошуковим компаніям в Інтернеті, таким як Google, аналізувати мільйони файлів на серверах по всьому світу. Ботів також можна використовувати як блокувальники електронної пошти або фільтрації ситуацій, коли потрібна швидка відповідь.

Життєвий цикл ботнету описаний на рисунку 2.3. Щоб стати частиною бот-мережі, бот має пройти цикл фаз, щоб скомпрометований хост був корисним за бажанням зловмисника. Ці фази можуть мати:

1. Початкове зараження: господар може бути заражений різними способами

- Шкідлива програма, здатна використовувати певну вразливість у хост-системі, щоб отримати привілеї користувача системи.

- Зловмисне програмне забезпечення автоматично завантажується під час відвідування шкідливих веб-сайтів.

- Зловмисне програмне забезпечення може бути доступне до хост-системи вкладеннями через електронну пошту

- Зловмисне програмне забезпечення може поширюватися через автозапуск USB.

2. Вторинна ін’єкція: зловмисне програмне забезпечення, завантажене під час початкового зараження, тепер завантажує оригінальну програму-бота та запускається на хост-машині, щоб стати активним хостом, підключившись до свого сервера. Бот можна завантажитися через FTP, HTTP та P2P.

3. Згуртування: коли бот успішно проникає на машину жертви, він зазвичай інформує) зловмисника про приєднання до ботнету.

4. Зловмисна активність: армія ботів отримує команди/інструкції від свого власника для здійснення шкідливих дій.

5. Технічне обслуговування та оновлення: боти потрібно періодично оновлювати, щоб вони стали ще більш стійкішими та непоміченішеми.

У сучасному середовищі ботнети майже завжди шкідливі, а останніми роками вони стали товаром, який можна орендувати або купити. Кіберзлочинці використовують бот-мережі для різноманітних зловмисних цілей, таких як зараження комп’ютерних систем, викрадення особистих даних, розсилка спаму тощо. Бот-мережа заражених комп’ютерів зазвичай контролюється «ботмайстром» або «пастухом ботів».

Ботнет може залишатися бездіяльним протягом місяців або років, перш ніж його активують або нададуть інструкцію здійснити атаку. Розробники ботнетів не використовують власні комп’ютери для розповсюдження шкідливих програм. Для виконання завдання вони використовують «ботів з дистанційним керуванням» або «комп’ютерів-зомбі».

Рисунок 2.3 – Життєвий цикл ботнету

7. «Зомбі-комп’ютери» — це комп’ютери, зламані хакером для виконання зловмисного завдання. Важливим і дуже руйнівним аспектом більшості ботнетів є їхня здатність виконувати розподілені атаки типу «відмова в обслуговуванні» (DDoS). DDoS-атака відбувається, коли кілька систем через адреси Інтернет-протоколу надсилають якомога більше запитів одному комп’ютеру чи службі в Інтернеті з наміром перевантажити їх і запобігти обслуговуванню законних запитів. DDoS-атаки часто призводять до повного збою веб-сайту на тривалий період часу. Крім того, стратегія здійснення атаки декількома системами, а не однією системою, призводить до майже миттєвого затоплення трафіку, спрямованого на цільовий комп’ютер або службу. Використання кількох комп’ютерних систем також заважає системним адміністраторам відстежувати, ідентифікувати та вимикати заражені машини, оскільки вхідний трафік походить із багатьох різних джерел.

8. Руткіти — це ПЗ, яке використовується для «приховування» ЗПЗ та ботнетів шляхом маскування того факту, що операційну систему комп'ютера було зламано. Руткіти дозволяють зловмисному програмному забезпеченню та ботнетам «ховатися», маскуючись під файли, які антивірусне програмне забезпечення пропускатиме. Руткіти відомі своєю здатністю надавати доступ до цільової системи через встановлення бекдору, щоб неавторизовані користувачі, такі як хакери, могли проникнути та виконувати свої шкідливі функції. Руткіти також відомі своєю здатністю приховувати своє існування. Зокрема, руткіти призначені для контролю над комп'ютерною системою таким чином, що навіть після того, як їх видалено, вони часто повертаються після перезавантаження системи. Крім того, вони можуть видалити будь-які сліди свого встановлення після того, як вони вторглися в систему. Оскільки їх важко виявити та видалити, руткіти представляють собою особливо неприємну форму зловмисного програмного забезпечення. Руткіти також можна писати та розробляти для легітимних і корисних функцій. Фактично, вони спочатку були розроблені як інструмент для адміністраторів комп'ютерних систем для моніторингу та керувати своїми системами. Зокрема, вони слугували бекдором для доступу системних адміністраторів у разі необхідності. Як і інші зловмисні програми, руткіти стають зловмисними лише тоді, коли вони створені з підлими намірами. Два основних типи руткітів — режим користувача та режим ядра. У режимі користувача, коли користувач комп'ютера отримує доступ до своєї робочої станції, зловмисне програмне забезпечення надає своєму власнику бекдор до комп'ютерної програми, яка працює в обліковому записі користувача. Однак у режимі ядра зловмисне програмне забезпечення працює в мережі як звичайне драйверне обладнання (наприклад, відеокарти, звукової карти, миші тощо). Оскільки руткіти режиму ядра виконуються на найвищому рівні доступу в операційній системі комп'ютера та центральному процесорі, вони вимагають

більш глибоких знань і навичок, тому їх складніше писати та створювати порівняно з руткітами режиму користувача.

9. Програмне забезпечення-вимагач — це зловмисне програмне забезпечення, призначене для запобігання доступу користувачів комп'ютерів до їхніх систем або вмісту файлів чи даних на жорсткому диску комп'ютера, якщо вони не здійснили платіж. Для створення зловмисного програмного забезпечення розробники програм-вимагачів зазвичай використовують процес, відомий як криптографія з відкритим ключем. Криптографія пов'язана з шифруванням, яке є процесом перетворення звичайної інформації в незрозумілу форму, і дешифруванням, яке є зворотним шифруванням або процесу перетворення незрозумілої інформації назад у звичайну форму.

Метод криптографії з відкритим ключем по суті передбачає створення пари ключів, відкритого ключа, який використовується для шифрування повідомлення (а також для перевірки того, що відправник повідомлення також є власником пари ключів) і закритого ключа. ключ, який використовується для розшифровки повідомлення. Як випливає з назви, зловмисне програмне забезпечення-вимагач вимагає «викупу» від користувачів комп'ютерів, перш ніж вони знову зможуть отримати доступ до своїх систем, файлів або даних. Жертви програм-вимагачів зазвичай отримують повідомлення з вимогою «викупу» або оплати в обмін на вимкнення шкідливого програмного забезпечення. Зокрема, коли програма-вимагач активована, вона шифрує цільові файли або дані, що зберігаються на локальних або підключених мережевих дисках, а потім відображає повідомлення, яке інформує користувача про те, що його або її файли/дані зашифровано, і якщо платіж не буде здійснено до вказаної дати , користувач втратить свої файли/дані (тобто закритий ключ для розшифровки інформації буде видалено). Важливо зауважити, що власник ЗПЗ зазвичай є єдиною особою, яка має доступ до закритого ключа, і ключ зазвичай зберігається на сервері контролера зловмисного програмного забезпечення.

10. Scareware – це тип зловмисного програмного забезпечення, яке використовує оманливі засоби, щоб викликати занепокоєння, шок або сприйняття загрози, щоб маніпулювати користувачами комп’ютерами, щоб вони купували небажане програмне забезпечення. Зокрема, цей тип зловмисного програмного забезпечення змушує користувачів повірити, що їхній комп’ютер заражено вірусом, а потім пропонує завантажити та заплатити за підроблене антивірусне програмне забезпечення, щоб видалити його. Фальшиве антивірусне програмне забезпечення упаковане таким чином, що воно нагадує законне програмне забезпечення безпеки, щоб ввести в оману споживачів. Одним із поширених типів відлякувальних програм є спливаючі рекламні вікна або банери, які відображають текстове повідомлення про те, що комп’ютер користувача, можливо, інфіковано шкідливим шпигунським програмним забезпеченням і його потрібно негайно видалити. Ці спливаючі вікна створені так, ніби вони надходять з операційної системи користувача, хоча насправді вони є веб-сторінкою. Ярлик «програмне забезпечення для лякання» також можна застосувати до шокуючих зображень, звуків або відео, які використовують пранкери, щоб викликати громадську тривогу чи паніку.

В таблиці 2 сформульовано загальні наслідки загрози на комп’ютерну систему.

Таблиця 2. Загальні ознаки зловмисної активності

№	Назва	Ознаки
1	2	3
1	Комп’ютерний вірус	Зменшення вільної пам'яті, уповільнення системи, затримки при запуску ПЗ, зміни в файлах, відбуваються часті збої у роботі системи;

Кінець таблиці 2

1	2	3
2	Комп’ютерний хробак	Низька продуктивність комп'ютера, зависання або збої системи, автоматичний запуск ПЗ, незвичайна поведінка системи, змінені файли, помилки ОС;
3	Троян	Маніпулювання браузером, повільна робота, зависання;
4	Шпигунське ПЗ	Маніпулювання браузером, часті збої роботи, поява невідомих файлів;
5	Кейлогер	Повільний браузер, затримка руху миші чи натискання клавіш або зникнення курсору;
6	Ботнет	Незрозуміла діяльність, повільний інтернет, повільне перезавантаження та завершення роботи системи;
7	Руткіт	Синій екран або збої роботи, дивна поведінка веб-браузера, повільна робота, налаштування Windows змінюються без дозволу, не належна робота веб-сторінок;
8	ПЗ-вимагач	Несанкціоноване шифрування даних, вилучення даних;
9	Scareware	Комп’ютер працює повільніше, маніпулювання браузером

2.3 Методи виявлення зловмисної програмного забезпечення в мережі

Виявлення зловмисного програмного забезпечення – це процес сканування зловмисного програмного забезпечення на вашому комп’ютері/смартфоні. Якщо систему заражено, потрібно виявити цю зловмисну активність перш ніж вона знищить комп’ютер. Проблеми під час завершення роботи чи перезапуску, часті збої системи чи повідомлення про помилки, електронні листи, які надсилаються автономно з вашого облікового запису, рішення безпеки вимкнено, підозрілі

файли швидкого доступу, батарея розряджається швидше, ніж очікувалося, незрозуміле використання даних, спливаюча реклама починає з'являтися скрізь у браузері тощо. це основні ознаки, які вказують на те, що ваш пристрій заражено шкідливим програмним забезпеченням. Сучасні комп'ютерні системи вже визначили достатню кількість методів для виявлення зловмисної активності:

1. Методи машинного навчання. Великою перевагою цих методів є те, що використання інструментів машинного навчання дозволяє виявляти невидимі сімейства зловмисного програмного забезпечення з дуже високою точністю та запам'ятовуванням. Цей метод забезпечує спосіб автоматизованого статичного аналізу коду та виявлення зловмисного програмного забезпечення з високою точністю та скорочує час, необхідний для аналізу активності мережі.

2. Багатошаровий персептрон (MLP) — це фаза ANN. MLP містить багато шарів, таких як вхідний шар, прихований шар і вихідний рівень. MLP широко використовується під наглядом навчання разом із зворотним алгоритмом для навчання мережі. Алгоритм зворотного поширення поєднує градієнт, щоб знайти правильну величину ваги для оновлення моделі мережі. Алгоритм містить завдання функції втрат для класифікації. Після навчання модель використовується для тестування зображень, які не є частиною навчального набору даних.

3. Технологія блокчейн – обмін даними на основі блокчейну та фреймворку виявлення спільноти. Для виявлення помилкових атак із впровадженням даних у систему використовується методологія Гільберта-Хуанга Трансформу та технологія реєстраційних облікових записів на основі блокчейну для посилення безпеки.

4. Марковська модель та глибоке навчання – метод класифікації зловмисного програмного забезпечення на рівні байтів на основі моделі маркова і глибокого навчання називається MDMC. Основним кроком у MDMC є перетворення двійкових файлів зловмисного програмного забезпечення на

маркери (Markov) шляхом перемикання матриці ймовірності передачі байтів. Після цього для класифікації маркових зображень використовується глибока згорточна нейронна мережа.

5. Фільтр спаму електронної пошти – спосіб фільтрації спаму та зловмисного програмного забезпечення в системі електронної пошти, включаючи стандартні рівні протоколів і політик.

6. Хмарні обчислення – метод, що представив хмарну структуру виявлення зловмисного програмного забезпечення, яка використовує гібридний метод для виявлення ЗПЗ. Хмарні обчислення відіграють важливу роль у всіх аспектах зберігання інформації та онлайнових послуг. Крім традиційної схеми зберігання та спільного використання, він має багато переваг, таких як легкий доступ, сховище запитів та зниження витрат. Використання цих технології може принести багато переваг для безпеки Інтернету речей (IoT), кіберфізичних систем (CPS) від різних типів кібератак. Хмарне середовище забезпечує велику обчислювальну потужність і дуже великі деталі виявлення зловмисного програмного забезпечення. Це також покращує продуктивність персонального обладнання виявлення, мобільних пристроїв і CPS.

7. Метод Smash – динамічний метод виявлення ЗПЗ на основі багатофункціонального ансамблевого навчання. По-перше, цей підхід використовує комбінацію функцій програмного забезпечення, таких як послідовності викликів API для високої точності виявлення та апаратних функцій низького рівня, таких як опір для уникнення дампу пам’яті і інструменти підвищення продуктивності АЗ. По-друге, він виберає високоякісну модель класифікатора для покращення виявлення однієї конкретної ознаки. І в завершенні він налаштовує інтегрований алгоритм навчання з кількома класифікаторами, що виявляють ЗПЗ та багато функцій, що можуть пояснити продуктивність такого ЗПЗ з багатьох вимірів для покращення ефективності виявлення.

8. Deep Belief або метод глибокої віри – підхід зосереджений на розробці ефективної обчислювальної системи на основі Deep BeliefNetworks для виявлення шкідливих програм. Цей фреймворк поєднує статичний аналіз високого рівня, динамічний аналіз і системні виклики для виділення функцій, щоб досягти найвищої точності.

9. Штучний інтелект – підхід, що визначає ефективність стратегій штучного інтелекту проти ризиків кібербезпеки.

10. Модель SCIRAS – цей метод представляє математичну модель для імітації зловмисного програмного забезпечення високого рівня. Зокрема, це модель світового класу SCIRAS (Susceptible-Carrier-Infectious-Recovered-Attacked-Susceptible), у якій розглядаються сприйнятливі пристрої, носії, заразні, набуті та атаковані пристрої. Це фрагментарна, вирішальна та глобальна модель, потужність якої базується на стандартній системі вимірювання. Для вивчення ефективності рішень використовує аналіз якості для різних рейтингів.

11. Евристичне виявлення – метод, що виконується за допомогою мережі викликів API із евристичним методом виявлення. Це призначено для визначення ефективності зловмисного програмного забезпечення, яке атакує мережу. Для перевірки шкідливих програм використовує кілька середовищ, які працюють як пісочниці. Цей метод дає підказки щодо захисту комп’ютерних систем, такі як використання антивірусного ПЗ, відмова у встановленні неавторизованих програм, доступ до небезпечних веб-сайтів, і необхідність встановлювати інші небажані програми.

12. Honeypot з машинним навчанням – спосіб змусити зловмисне програмне забезпечення використовувати honeypot за допомогою машинного навчання. Honeypot можна використовувати як пастку для підозрілих пакетів, тоді як машинне навчання може виявляти зловмисне програмне забезпечення шляхом класифікації класів.

13. Багаторівнева безпека – підхід, що базується на багаторівневій програмній системі безпеки та захисту від програм-вимагачів. Цей підхід запроваджує антивірусне програмне забезпечення для локальних машин, добре налаштовані брандмауери, ефективну DNS/веб-фільтрацію, захист електронної пошти, резервне копіювання та навчання персоналу.

Висновки до другого розділу

Поширення та еволюція шкідливого програмного забезпечення є серйозною загрозою не лише для комп'ютерних систем і мобільних пристроїв, а й для користувачів комп'ютерів. На сьогоднішній день існує чимало досліджень щодо виявлення, класифікації та запобігання зловмисному програмному забезпеченню. У даному розділі було розглянути та формалізовано зразки зловмисного програмного забезпечення, способи їх поширення та наслідки для роботи системи. Крім того досліджено методи, які сучасні методи захисту використовують для виявлення ЗПЗ та зловмисної активності. Вивчення та розуміння того, які загрози можуть впливати на комп'ютерну систему, та існуючих методів є виправданим, оскільки такі висновки є доречними та вирішальними для розробки практичних і ефективних стратегій і політики боротьби з шкідливим ПЗ. Для побудови системи буде доречно використати один найбільш точних і ефективних методів – метод машинного навчання.

3 АРХІТЕКТУРА САМООРГАНІЗОВАНОЇ СИСТЕМИ ПРОГНОЗУВАННЯ

3.1 Архітектура системи

Для забезпечення захищеності системи необхідно визначити три важливі проблеми, які найчастіше пов'язують з безпекою мережі.

1. Перша проблема пов’язується зі швидким збільшенням кількості мережевих даних. Таке зростання насамперед пов’язано з використанням Інтернету речей (IoT), хмарних сервісів та постійного зростання мережі пристроїв. Покращення методів аналізу даних має включати збільшення швидкості та надійності процесів аналізу.

2. Друга проблема виявляє себе в тому, що точність відстеження та інтерпретація сильно підвищує якість результатів. Аналіз вимагає більш контекстно-специфічних спостережень, що підкреслюють абстрактні висновки та спостереження більшого рівня. Усі зміни повинні бути відстежуваними до конкретного користувача, версії ОС або протоколів окремих програм.

3. Третя проблема сучасних мереж — це збільшення різноманітності протоколів і досить масивна передача використовуваних даних у сучасних мережах. В такому випадку, виражається надзвичайно високий рівень складності, коли потрібно відрізнити зловмисний трафік від звичайного нормального. Це покращує ймовірність ненадійних даних та збільшує потенціальність впливу вразливостей нульового дня.

Статистика показує, що найбільше методів виявлення зловмисних дій є методами заснованими на правилах і методами прогнозованого моделювання (виявлення аномалій).

Для правильного функціонування системи роботи, спрямованого на визначення зловмисної активності, потрібно визначити потоки даних трафіку. Ці потоки зображені на рисунку 3.1.

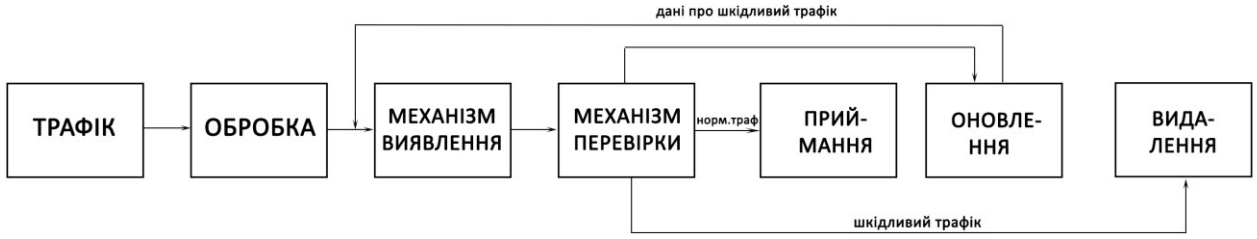


Рисунок 3.1 – Інформаційні потоки для споектованої системи прогнозування

На рисунку 3.2 зображено інформаційні потоки системи в тій частині у якій розглядається трафік на предмет класифікації:

Спочатку вхідний трафік який був зібраний системою подається на модуль попередньої обробки. Даний модуль фіксує і обробляє вхідний трафік в системі реального часу. Перехоплені TCP з’єднання опрацьовуються для вилучення відповідних функцій. Надалі ці функції стають доступними як вхідний вектор для механізму виявлення – FG-SOINN.

Для виконання дослідження перевірка була виконана використовуючи назви 41-го атрибуту набору даних NSL-KDD. Його структура базується на поясненні того, що характеристики з’єднання TCP мають вказувати на те, чи з’єднання визначене як шкідливе або ж ні. Якщо з’єднання визначається як атака, то система робить висновок, що воно є шкідливим і видаляє його.

Далі інформація про зловмисне з’єднання подається на механізм виявлення зловмисної активності, який після опрацювання даних передає результуючу інформацію про трафік на модуль перевірки даних.

Далі цей модуль перевірки визначає зміну точності системи та її покращення шляхом підтвердження передбаченої мітки. Саме цей етап відповідає за розподілення трафіку на шкідливий або нормальний. Якщо відбулося підтвердження того, що вхідний трафік є нормальним він передається користувачу для подальшої роботи. Але ж якщо системою відбулося підтвердження про виявлення шкідливого трафіку, власне цей модуль перевірки даних надсилає всі невдалі прогнози з попереднього модуля на модуль оновлення і видаляє з набору даних трафіку.

Модуль оновлення системи працює у дві фази: фаза активна та фаза оновлення. На активній фазі модуль приймає рішення використовуючи дані якими він володіє в той час, і на фазі оновлення системи, коли модуль оновлює дані системи за допомогою невдалих прогнозів для покращення її майбутніх можливостей. Фази будуть виконуватися паралельно, коли це необхідно, або чергуватися, відповідно до даних мережевого трафіку.

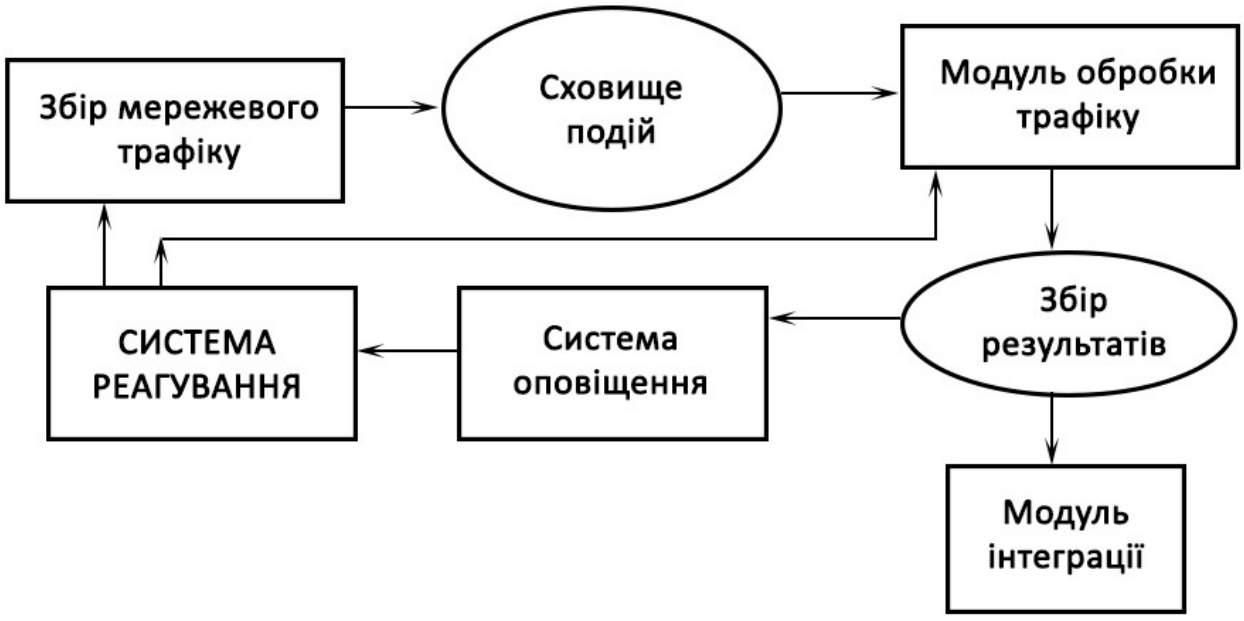


Рисунок 3.2 – Архітектура самоорганізованої системи

Архітектура системи буде складатися з наступних компонент:

1. Збір трафіку мережі який необхідний для використання всієї наявної інформації про пристрої, що використовуються у мережі. Крім того, він виконуватиме функцію перетворення вихідного мережевого трафіку в потрібному вигляді (для обрахування необхідних параметрів) і запис даних в сховище подій.

2. Сховище подій дана система визначає як місце для збереження інформації, що потім аналізується системою на наявність зловмисного трафіку і мережевих атак. Кожен пункт записів у сховищі зберігатиме інформацію про потоки даних за параметрами, що необхідні для модуля обробки трафіку.

3. Модуль обробки трафіку потрібен як система, що працює на алгоритмі SOINN (самоорганізованих інкрементних нейронних мереж). Даних модуль виконуватиме аналіз трафіку всіх записів про потоки, які зберігатимуться в базі даних вихідних подій з використанням алгоритмів, які основані на методах машинного навчання. В кінці для кожного запису модуль визначає тип з'єднання: нормальне або зловмисне, а також вид атаки у випадку її виявлення.

4. Результати аналізу даних будуть занотовані в базу даних результатів. Місце збору результатів представлене як відокремлена база даних, яка використовується для збереження виявлених аномалій.

5. Дані мають використовуватися системою сповіщень, а також модулем сумісності для вилучення та подальшого використання результатів аналізу.

6. Система реагування виконуватиме функції «сміттяра». Кожен запис, який був визначений системою як шкідливий буде знищуватися, не визначені записи, тобто такі записи, які не є і не нормальними і не шкідливими, будуть повернуті в модуль обробки трафіку. Потім дана інформація буде використовуватись для подальшої повторної обробки. Після того, система знову повертається до модуля збору мережевого трафіку який продовжує цикл роботи.

7. Модуль інтеграції являє собою API для можливості інтегрування з системами реагування, інтерфейс системної взаємодії за допомогою http-запитів.

3.2 Підсистеми прогнозування

Схеми прогнозованого керування розподіленою моделлю (DMPC) стали популярним вибором для проблем мережевого керування. Згідно з цим підходом, локальні контролери використовують модель для прогнозування поведінки своєї підсистеми протягом певного горизонту, щоб знайти послідовність вхідних даних, яка оптимізує її еволюцію відповідно до заданого критерію. Деякими зручними функціями цього методу є явна обробка обмежень та обмін інформацією між контролерами для координації їх активаціїі мінімізувати небажані взаємні взаємодії.

Останніми роками спостерігається зростаючий інтерес до розподілених методів керування через їх чудову масштабованість у великомасштабних програмах, таких як інтелектуальні мережі. Цей підхід розглядає загальну систему як сукупність менших частин , тобто підсистем, якими локально керують блоки керування. Серед багатьох підходів до розподіленого управління ми особливо зацікавлені в передбачуваному керуванні розподіленою моделлю , яка надає кілька переваг для мережевих програм керування. Зокрема, прогностичні контролери використовують модель системи, щоб передбачити її еволюцію на заданому горизонті та побудувати задачу оптимізації, для знаходження найбільш відповідних послідовностей керуючих дій відповідно до заданого критерію. Будучи комп’ютерним підходом, метод постійно збільшує розмір проблем, які він може вирішити завдяки прогресу в інформаційних і комунікаційних технологіях. Іншою чудовою характеристикою цього сімейства є наявність методів, доступних для контролерів кластерів

Критичною проблемою розподілених методів є координація рішень агентів. Загальновідомо, що ставлення контролерів, які можуть не бажати співпрацювати, та їхні знання про загальну систему мають великий вплив на місцеві рішення, а отже, і на глобальну продуктивність. З цією метою вони взаємодіють зі своїм фізичним середовищем, передають локальні дані та оновлюють інформацію про сусідів, отже, вимагаючи підключення, надійності та безпеки свого локального обладнання та мережі зв'язку. Стосовно останнього часто визначають три різні цілі безпеки: конфіденційність, тобто збереження важливих даних у таємниці, цілісність, тобто гарантування вірності даних, і доступність, тобто забезпечення доступності даних у потрібний час.

Система самонавчання є одним з ефективних методів боротьби з сучасними атаками. Вона використовує контрольовані, напівконтрольовані та неконтрольовані механізми машинного навчання для вивчення шаблонів різних звичайних і зловмисних дій за допомогою великого корпусу звичайних і атакуючих мережевих подій і подій на рівні хоста. Існуючі рішення, засновані на машинному навчанні, видають високий рівень хибних спрацьовувань із високою обчислювальною вартістю. Це тому, що класифікатори машинного навчання вивчають характеристики простих функцій TCP/IP локально. Глибоке навчання — це складна підмережа машинного навчання, яка вивчає ієрархічні представлення функцій і приховані послідовні зв'язки, передаючи інформацію TCP/IP на кількох прихованих рівнях.

Безперервне навчання є довгостроковою метою машинного навчання. Цю парадигму навчання також називають поступовим навчанням або ж навчанням протягом усього життя, яке нагадує те, як люди та інші тварини набувають і вдосконалюють свої знання. Насправді відмінною рисою людського інтелекту є здатність застосовувати знання з однієї області для вирішення проблем в іншій області. Таке перехідне навчання також часто розглядається як бажану властивість системи безперервного навчання.

Самоорганізуючі інкрементні нейронні мережі (SOINN) охоплюють сімейство нейронних мереж, спільним для яких є те, що вони знаходять топологічне відображення вхідних даних у мережеву структуру шляхом конкурентного навчання. SOINN відображає p -вимірний вхід $x = x_{1,}x_2 \dots x_p$ де $x_i \in i^{-M}$ значенням ознаки окремого вузла неорієнтованого графу. Відображення відповідає точці в p - вимірному просторі ознак. Навчання в SOINN означає адаптацію топологічної карти: вузли можуть переміщатися, об'єднуватися з іншими вузлами, залишатися одиночними або видалятися, а межі між вузлами можна створювати або видаляти. Вузол розглядається як мікрокластер вхідних випадків, що знаходяться близько один від одного. Ребра можна розглядаються як консолідовані зв'язки з пов'язаними вузлами, наприклад, вузлами, які належать одному (макро)кластеру.

Методи використання алгоритмів SOINN:

1. Неконтрольоване навчання. Оригінальний SOINN найчастіше вживаються для неконтрольованого навчання. SOINN використовується для вивчення топологічної структури вхідних даних, він здатний поступово зростати та враховувати шаблони введення нестаціонарного розподілу даних. Він може розділяти класи з перекриттям низької щільності та виявляти основну структуру кластерів, які забруднені шумом. Він автоматично вивчає кількість вузлів і структуру мережі, повідомляє кількість кластерів і дає типові прототипи кожного кластера.

2. Кероване навчання використовує SOINN для кожного класу окремо та генерує типові прототипи. Цей метод автоматично вивчає кількість прототипів, необхідних для визначення межі рішення. Для різних класів вивчені прототипи можуть бути різними. Він стійкий до зашумлених навчальних даних і реалізував дуже швидку класифікацію.

3. Напівконтрольоване навчання. Використовуючи позначені дані та велику кількість немаркованих даних, запропонований метод автоматично вивчає топологію розподілу вхідних даних без попереднього знання, згодом він позначає всі згенеровані вузли та ділить вивчену структуру топології на підструктури або відповідаючи на заняття. Ваги вузлів використовуються як прототипи векторів для реалізації класифікації. Під час навчання нові позначені чи немарковані дані можуть поступово додаватися до системи.

4. Активне навчання пропонує онлайн-алгоритм поетапного активного навчання на основі SOINN. Він використовує SOINN для представлення топологічної структури вхідних даних, а потім розділяє згенеровані вузли на різні групи та підкластери. Потім він активно позначає деякі вузли і використовує щоб позначити всі непомічені вузли. Він запитує мітки деяких важливих зразків, а не вибирає позначені зразки випадковим чином. Автоматично вивчає кількість вузлів і векторів, необхідних для поточного завдання. Експерименти з використанням штучних даних і даних реального світу показують, що запропонований метод працює ефективно та результативно.

5. Асоціативна пам'ять, що працює в реальному середовищі, повинна добре працювати в онлайн-поступовому навчанні та бути стійкою до зашумлених даних, оскільки зашумлені асоціативні шаблони представлені послідовно в реальному середовищі. Нові асоціативні пари, які подаються послідовно, можна точно засвоїти, не забуваючи раніше вивчені шаблони. Розмір пам'яті методу адаптивно збільшується разом із шаблонами навчання. Таким чином, він не страждає ні від надмірності, ні від недостатнього обсягу пам’яті, навіть у середовищі, в якому максимальна кількість асоціативних пар, які мають бути представлені, невідома до початку навчання. Запропонована асоціативна пам’ять працює як двонаправлена асоціативна пам’ять «один-до-багатьох» або «багато-до-одного» та має справу не лише з біполярними даними, а й з реальними даними.

6. Обґрунтування на основі шаблонів. Обробляючи шаблони як вектори з дійсними значеннями та класифікуючи подібні правила «якщо-тоді» в кластери в довгостроковій пам’яті, запропонований метод може зберігати засновані на шаблонах правила «якщо-тоді» пропозиційної логіки, включаючи кон’юнкції, з’єднання та заперечення. Він також досягає деяких важливих властивостей для інтелектуальних систем, таких як поступове навчання, узагальнення, уникнення дублювання результатів і стійкість до шуму. Експерименти показують, що запропонований метод дуже ефективний для інтелектуальних систем, які автономно вирішують різноманітні завдання в реальному середовищі.

3.3 Підсистема прийняття рішень в самоорганізованій системі

Самоорганізація підсистеми передбачає можливість членів цієї підсистеми приймати рішення самостійно. Для самоорганізованих груп керівник групи не вирішує щодо завдань, які повинен виконувати кожен член групи. Швидше, група шляхом самокоординації призначатиме завдання своїм членам. Самоорганізація та утономія членів у групі відповідають один одному. Формалізація пов’язана зі ступенем впровадження загальних правил і формальної поведінки в організації. Високий ступінь формалізації має сенс переважно в статичних середовищах. У випадку складних динамічних середовищ необхідний високий ступінь гнучкості.

Самоорганізована та інкрементна нейронна мережа (SOINN) — це механізм неконтрольованого навчання для немаркованих даних. SOINN вже використовувався в інших дослідження як метод кластеризації, який обробляє контрольовані дані. SOINN пропонує навчання без вчителя для інкрементного методу кластеризації з відносно високою швидкістю обробки при низькій вартості обчислення. Крім того, складність і розмір мережі SOINN контролюються та стабілізуються через «сміття» або відкридання непотрібних вузлів (нейронів).

Техніка колектора або методика зношеності вузла, визначає коли і після якого вузли періоду його буде видалено. Але якщо вони не оновлюються в заданий час вони просто видаляються. Ця властивість робить його привабливим для динаміки будь-якого середовища, де потрібне тривале навчання. Для забезпечити його масштабованість при розширенні зростанні мережа керується параметром n, де кілька пар SOINN будуть використовуватися для методу контрольованої кластеризації. SOINN ініціалізує мережу за допомогою порожніх наборів вузлів, а потім додає перші два вузли до списку, із ваговими векторами, встановленими як два вхідні вектори. Далі нейронна мережа для кожного вхідного вектора знаходить найближчий вузол (переможець) і другий найближчий вузол (другий переможець) вектора шляхом вимірювання відстані S1 і S2 [81] від кожного входу до кожного вузла з рівняннями (1) і (2):

$$s_1 = argmin_{C \in A} dist(x, w_c) \quad (1)$$

$$s_2 = argmin_{C \in A - \{s_1\}} dist(x, w_c) \quad (2)$$

Якщо ж вхідний вектор відноситься до того ж кластеру, що й переможний вузол або другий переможний вузол на основі відстаней, обрахованих пороговим критерієм подібності, алгоритм оновлює ваговий вектор вузла та його сусідів з вектором ваговим значення входу та поєднує його вузол з ребром. Якщо вектор входу не належить до того ж кластера переможного вузла або другого переможного вузла, механізм додає інший (новий) вузол до мережі.

Ключовою характеристикою мережі SOINN є її адаптивність до змін розподілу ймовірностей невідомого процесу генерації даних. Для виконання завдання прогнозування зловмисної активності пропонується модифікувати наявний алгоритм SOINN, який успадкує деякі ідеї, але принципово відрізняється не буде. У попередніх поколіннях SOINN пакети вузлів і ребер очищаються через

фіксовані, визначені користувачем проміжки часу, що призводить до раптових змін у структурі мережі, оскільки великі регіони мережі періодично видаляються перед відновленням навчання. Це «пакетне забування» через фіксовані проміжки часу конфліктує з більш поступовим забуванням, яке зазвичай можна спостерігати в природних когнітивних системах.

У SOINN видалення вузлів і ребер визначається двома параметрами, які потрібно оптимізувати для кожної наявної програми за допомогою перехресної перевірки або подібних підходів повторної вибірки. FG-SOINN усуває цей недолік, розглядаючи видалення вузлів і ребер як невід'ємну частину процесу навчання. Мережа видаляє вузли та ребра, якщо вони не відповідають поточному навчальному завданню.

Висновки до третього розділу

В ході роботи над третім розділом було побудовано архітектуру мережі яка задовольнятиме задачу дослідження. Крім того, було розглянути системи і підсистеми прийняття рішень в комп'ютерних мережах. Така архітектура буде працювати на основі системи виявлення вторгнень яка буде складатися з наступних компонент: Збір трафіку мережі, сховища подій, модуля обробки трафіку, модуля збору результатів, системи сповіщення, реагування та методу інтеграції. Для компоненти модуля обробки трафіку буде розроблено модифікований алгоритм інкрементного самоорганізованого навчання FG-SOINN, який буде використовувати розглянутий раніше метод неконтрольованого навчання. Такий алгоритм буде ефективніший ніж звичайний SOINN.

4 ДОСЛІДЖЕННЯ РЕАЛІЗОВАНОЇ САМООГРАНІЗОВАНОЇ СИСТЕМИ

4.1 Підготовка даних

Дані, які використовуються для навчання класифікатора, взяті з набору даних NSL-KDD, наданого Канадським інститутом кібербезпеки. Він був створений шляхом охоплення мережевого трафіку протягом десяти днів роботи в контрольованому мережевому середовищі на AWS, де проводився реалістичний трафік та різні сценарії атак. У результаті набір даних містить як безпечний мережевий трафік, так і дані про найпоширеніші мережеві атаки.

Набір даних складається з необроблених записів мережі у форматі pcap, а також файлів csv, створених за допомогою CICFlowMeter-V3, які містять 80 статистичних даних окремих мережевих потоків у поєднанні з відповідними мітками (рис. 4.1).

1	duration,protocol_type,service,flag,src_bytes,dst_bytes,attack,wrong_fragment,urgent,hot,num_failed_logins,logged_in,num_
2	0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,229,10,0,0,1,1,0.04,0.06,0,255,10,0.04,0.06,0,0,0,0,1,1,attack
3	0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,136,1,0,0,1,1,0.01,0.06,0,255,1,0,0.06,0,0,0,0,1,1,attack
4	2,tcp,ftp_data,SF,12983,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,134,86,0.61,0.04,0.61,0.02,0,0,0,0,normal
5	0,icmp,eco_i,SF,20,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,65,0,0,0,0,1,0,1,3,57,1,0,1,0.28,0,0,0,0,attack
6	1,tcp,telnet,RSTO,0,15,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,8,0,0.12,1,0.5,1,0,0.75,29,86,0.31,0.17,0.03,0.02,0,0,0.83,0.71,attack
7	0,tcp,http,SF,267,14515,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,4,4,0,0,0,0,1,0,0,155,255,1,0,0.01,0.03,0.01,0,0,0,normal
8	0,tcp,smtp,SF,1022,387,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,3,0,0,0,0,1,0,1,255,28,0.11,0.72,0,0,0,0,0.72,0.04,normal
9	0,tcp,telnet,SF,129,174,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,255,255,1,0,0,0,0.01,0.01,0.02,0.02,attack
10	0,tcp,http,SF,327,467,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,33,47,0,0,0,0,1,0,0.04,151,255,1,0,0.01,0.03,0,0,0,0,normal
11	0,tcp,ftp,SF,26,157,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,1,0,0,52,26,0.5,0.08,0.02,0,0,0,0,0,attack
12	0,tcp,telnet,SF,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,255,128,0.5,0.01,0,0,0,0,0.66,0.32,attack
13	0,tcp,smtp,SF,616,330,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,2,0,0,0,0,1,0,1,255,129,0.51,0.03,0,0,0,0,0.33,0,normal
14	0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,111,2,0,0,1,1,0.02,0.07,0,255,2,0.01,0.07,0,0,0,0,1,1,attack
15	0,tcp,telnet,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,120,120,1,1,0,0,1,0,0,235,171,0.73,0.07,0,0,0.69,0.95,0.02,0,attack
16	37,tcp,telnet,SF,773,364200,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,1,0,0,38,73,0.16,0.05,0.03,0.04,0,0.77,0,0.07,normal
17	0,tcp,http,SF,350,3610,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,8,8,0,0,0,0,1,0,0,71,255,1,0,0.01,0.04,0,0,0,0,normal
18	0,tcp,http,SF,213,659,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,24,24,0,0,0,0,1,0,0,255,255,1,0,0,0,0,0,0,0,normal
19	0,tcp,http,SF,246,2090,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,16,16,0,0,0,0,1,0,0,35,255,1,0,0.03,0.05,0,0,0,0,normal
20	0,udp,private,SF,45,44,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,505,505,0,0,0,0,1,0,0,255,255,1,0,1,0,0,0,0,0,normal
21	0,tcp,private,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,204,18,0,0,1,1,0.09,0.07,0,255,18,0.07,0.07,0,0,0,0,1,1,attack

Рисунок 4.1 – Набір даних NSL-KDD для тренування системи

Мережевий потік визначається як сукупність взаємопов’язаних мережевих пакетів, ідентифікованих такими властивостями:

1. Джерело IP
2. IP призначення
3. Порт джерела
4. Порт призначення
5. Протокол

Набір даних містить приблизно 16 мільйонів окремих мережевих потоків і охоплює такі сценарії атак (рис. 4.2):

1. DOS
2. Нормальний трафік
3. Атаки зондування
4. R2L
5. U2R

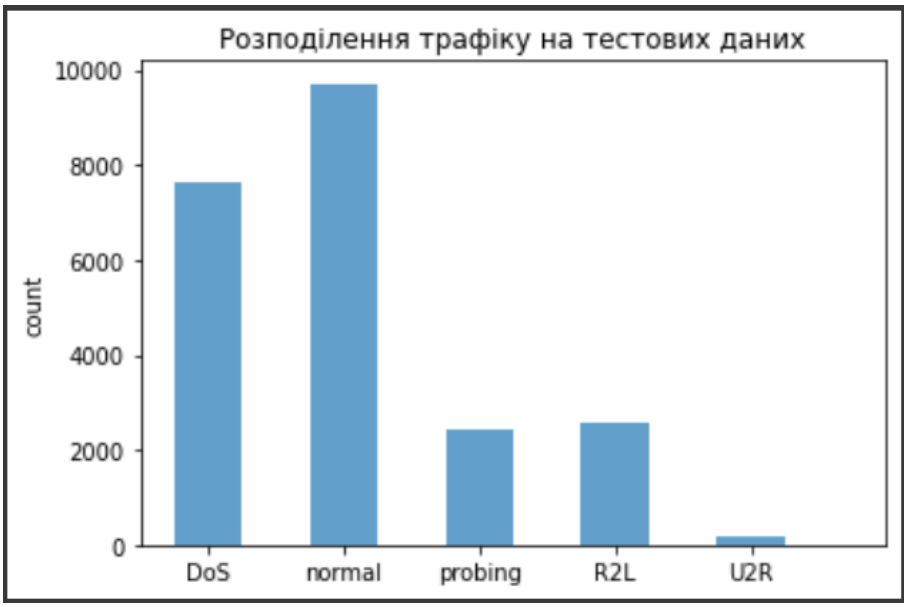


Рисунок 4.2 – Розподілення трафіку на тестових даних

- Атаки типу DoS – атака відмови в обслуговуванні. Раніше згадувалась у роботі. Тип атаки при якій з комп’ютера на системні ресурси подається велика кількість запитів з метою доведення системи до відмови, тобто створення умов при яких звичайні користувачі не можу отримати доступ до системи і відповідно до її даних. Наприклад: атаки pod, smurf, teardrop, apache2, mailbomb.

- Нормальний трафік - «робочий» мережевий трафік який не несе шкоди системі.

- Probing attack або атаки зондування - є стартовою фазою атаки на веб-ресурси та веб-додатки. Під час такої атаки зловмисник виконує збір інформації про структурні особливості веб-додатку (сторінки, параметри тощо) та додатково допоміжну інфраструктуру (операційну систему, бази даних тощо). Веб-сайти перевіряються на конкретизовані вразливості в ПЗ інфраструктури (наприклад, IIS), і також невизначені вразливості в окремому спеціальному коді, який розроблений для цільової конкретної програми. Приклад атаки: ipsweep, nmap, portsweep, satan, mscan, saint, worm.

- Remote to user (R2L) — це один із типів атак комп’ютерної мережі, під час яких зловмисник надсилає набір пакетів на інший комп’ютер або сервер через мережу, до якої він/вона не має дозволу на доступ як локальний користувач. Атака R2L на з’єднання містять не лише мережевий рівень, а й функції на рівні хоста, і тому здебільшого їх досить важко їх. Приклад: атаки sendmail, xlock, xsnoop.

- User to Root (U2R): спочатку зловмисник отримує доступ до звичайного облікового запису користувача, пізніше отримує доступ до root, використовуючи вразливі місця системи. Приклади: атаки Perl, Load Module і Eject.

4.2 Програмна реалізація алгоритму прийняття рішень FG-SOINN самоорганізованої системи для прогнозування зловмисної активності

Спершу для повного функціоналу системи потрібно програмно реалізувати модифікований алгоритм SOINN який названо FG-SOINN (Forgetting Garbage). Для початку треба визначити атрибути і параметри алгоритму:

Параметри:

- x1, x2, x3 – приклади для випадкової ініціалізації;
- max_edge_age - максимальний вік вузла;
- кожна ітерація iter_lambda запускає процес групування та процес видалення вузла;
- pull_factor – коефіцієнт тяги для об’єднання вузлів.

Атрибути:

- t – лічильник ітерацій оновлення мережі. Збільшується кожного разу, коли вхідний сигнал обробляється;
- pull_factor – коефіцієнт тяги для об’єднання вузлів;
- Igraph – графік igraph, що представляє інкрементну нейронну мережу.

Вузли мають такі атрибути:

- w – ваги вузла;
- wt - час необхідний для перемоги вузла;
- st - поріг подібності вузла;
- it – час простою вузла, тобто лічильник того, скільки ітерацій вузлів не було обрано як переможець;
- u – корисність вузла;
- c – остаточна мітка класу вузла;
- cl – список міток класів, призначених для вузла.

Ребра мають такі атрибути:

- it - час простою ребра;

- wt - кількість разів коли ребро було скинуто.

Крім того алгоритм буде реалізовувати такі функції:

- distance - обчислює часткову відстань між двома масивами a і b (рис.4.3)

```
def _distance(self, a, b):
    f = 0.5
    diff = abs(a - b) ** f
    sum = np.sum(diff)
    distance = math.pow(sum, 1/f)
    return distance
```

Рисунок 4.3 – Функція distance

- get_n1_n2 – обчислює переможця (найближчий вузол n1 як вершина igraph) та другого переможця (другий найближчий вузол n2 як вершина igraph)(рис. 4.4)

```
def _get_n1_n2(self, x):
    distances = dict()
    for n in self.network.vs:
        distances[n] = self._distance(x, n['w'])
    n1 = min(distances, key=distances.get)
    del distances[n1]
    n2 = min(distances, key=distances.get)
    return n1, n2
```

Рисунок 4.4 – Функція get_n1_n2

- similarity_threshold - Обчислює порогові значення подібності переможця та другого переможця. Якщо переможець (або другий переможець) є єдиним вузлом, тоді поріг відповідатиме відстані до найближчого вузла. Якщо переможець (або другий переможець) не одиночний, то поріг відповідатиме відстані до найдальшого сусіда (рис. 4.6).

```
def _similarity_threshold(self, x, node):
    distances = []
    if node.degree() == 0:
        for n in self.network.vs:
            if n.index != node.index:
                distances.append(self._distance(node['w'], n['w']))
        d = min(distances)
    else:
        for n in node.neighbors():
            distances.append(self._distance(node['w'], n['w']))
        d = max(distances)
    if d != 0:
        d += d * (1 - 1 / node['wt'])
    node['st'] = d
```

Рисунок 4.6 – Функція get_n1_n2

- add_node – функція додавання нового вузла до мережі (рис. 4.7)

```
def _add_node(self, weights, y):
    new_node = self.network.add_vertex()
    new_node['w'] = weights
    new_node['wt'] = 1
    new_node['it'] = 1
    new_node['cl'] = [y]
    new_node['c'] = y
```

Рисунок 4.7 – Функція add_node

- merge_nodes - об’єднання вузлів-переможців з новим вхідним сигналом x (масивом ваги вузла), щоб вузол-переможець і його сусіди були налаштовані відповідно до нового випадку x. Даний зсув пропорційний wt вузла-переможця, чим більший wt, тим менший вплив має x на решту мережі (рис. 4.8).

```
def _merge_nodes(self, n1, x):
    n1['w'] = n1['w'] + ((x - n1['w']) / n1['wt'])
    for n in n1.neighbors():
        pulled_weight = self.pull_factor * n['wt']
        n['w'] = n['w'] + ((x - n['w']) / pulled_weight)
    n1['it'] = 1
```

Рисунок 4.8 – Функція add_node

- linking – унікальна функція алгоритму FG-SOINN, що намагається зв’язати вузли, які ймовірно представляють сигнал, а не шум. Зв’язування залежить від надійності вузла (рис. 4.9).

```
def _linking(self, n1, n2):
    n_edges = self.network.ecount()
    if n_edges == 0 or not self.network.are_connected(n1.index, n2.index):
        edge = self.network.add_edge(source=n1.index, target=n2.index)
        edge['it'] = 1
    else:
        self.network.es[self.network.get_eid(n1.index, n2.index)]['it'] = 0
    for e in self.network.vs[n1.index].incident():
        e['it'] += 1
```

Рисунок 4.9 – Функція linking

- edge_deletion – дана функція видаляє ребра які являються застарілими або з’єднує різні кластери (рис. 4.10)

```
def _edge_deletion(self):
    for e in self.network.es:
        source = self.network.vs[e.source]
        target = self.network.vs[e.target]
        if e['it'] > self.max_edge_age or source['c'] != target['c']:
            self.network.delete_edges(e.index)
            self.n_del_edges += 1
```

Рисунок 4.10 – Функція edge_deletion

- nodes_deletion – функція видалення вузлів на основі оцінки їх корисності. Високі порогові значення показують, що обраний вузол найрідше обирався як переможний (рис. 4.11)

```
def _nodes_deletion(self):
    max_u = 0.0
    us = []
    for n in self.network.vs:
        n['u'] = n['wt'] / n['it']
        us.append(n['u'])
        if n['u'] > max_u:
            max_u = n['u']
    u_mean = np.mean(us)
    for n in self.network.vs:
        if n.degree() == 0 and n['u'] < u_mean:
            prob_survival = n['u'] / max_u
            prob_deletion = 1 - prob_survival
            if prob_deletion > prob_survival:
                self.network.delete_vertices(n.index)
                self.n_del_nodes += 1
    for n in self.network.vs:
        n['it'] += 1
```

Рисунок 4.11 – Функція nodes_deletion

- group – функція визначає мітки класів, вибравши клас, який найчастіше призначається для кожного вузла (рис. 4.12)

```
def _group(self):
    for n in self.network.vs:
        occurence_count = Counter(n['cl'])
        n['c'] = occurence_count.most_common(1)[0][0]
        n['cl'] = [n['c']]
```

Рисунок 4.12 – Функція group

- input_signal – функція для підбору вхідного сигналу. Якщо мітка не встановлена, то даний сигнал вважається зашумленим (рис. 4.13).

Його параметри:

x – вектор ваги вхідного класу

y – клас або мітка пов’язана з вхідним сигналом

learning – встановлення True – навчання, False – передбачення

повертає параметр prediction та confidence який означає прогнозований ярлик та ймовірність істинності результатів відповідно.

```
def input_signal(self, x, y=None, learning=True):
    n1, n2 = self._get_n1_n2(x)
    if learning:
        if y is None:
            y = NOISE_LABEL
        self.t += 1
        n_nodes = self.network.vcount()
        n_edges = self.network.ecount()
        prediction = n1['c']
        self._similarity_threshold(x, n1)
        self._similarity_threshold(x, n2)
        d1 = self._distance(x, n1['w'])
        d2 = self._distance(x, n2['w'])
        if d1 >= n1['st'] or d2 >= n2['st']:
            self._add_node(x, y)
        else:
            n1['wt'] += 1
            if y != NOISE_LABEL:
                n1['cl'].append(y)
            self._merge_nodes(n1, x)
            self._linking(n1, n2)
        if self.t % self.iter_lambda == 0:
            if n_nodes > 3:
                self._nodes_deletion()
            self._group()
            if n_edges > 3:
                self._edge_deletion()
    else:
        prediction = n1['c']
    confidence = 0
    if n1 in self.network.vs:
        self._similarity_threshold(x, n1)
        if n1['st'] != None and n1['st'] != 0:
            confidence = 1 - self._distance(
                x, n1['w']) / n1['st']

    return prediction, confidence
```

Рисунок 4.13 – Функція input_signal

Основна концепція запропонованого алгоритму полягає в поступовому створенні механізму захисту мережі. На початковому етапі механізм виявлення навчається, використовуючи відносно невелику вибірку мережевих даних, достатню для базового захисту мережі. Згодом, коли стає доступним більше мережевих даних, механізм поступово оновлюється вхідними даними класів, які йому не вдалося виявити, щоб удосконалити та розширити свої захисні

можливості. Щоб підтримати процес навчання, механізм перевірки вирішує, чи не вдалося прийняти рішення. Щоб розширити свої можливості, основний механізм виявлення повинен бути в змозі класифікувати мережеві дані за багатьма класами, не тільки про те, чи є з'єднання атакою чи ні, але й про тип атаки.

Програмна реалізація алгоритму FG-SOINN була розроблена з допомогою мови програмування Pethon та середовища Jupiter Notebook. Повний лістинг показаний в додатку А.

4.3 Програмна реалізація самооганізованої системи розпізнавання зловмисної активності

Програмна реалізація самоорганізованої системи IDS відбувається з використанням моделі визначеної у попередньому розділі, а саме FG-SOINN.

Спочатку потрібно визначити допоміжні функції системи:

1. Тренування – функція в якій дані вписуються в систему. Визначається декількома параметрами:

- Модель - FG-SOINN

- Дані – вхідний набір даних з нормалізованими ознаками

- Мітки пов'язані з прикладами у вхідному наборі

2. Оновлення – функція яка виконує прогнозування.

Неправильні прогнози повертаються. Виводить показники ефективності. Також у цій функції визначаються деякі метрики оцінки:

- Справжній позитивний коефіцієнт (TPR) або коефіцієнт виявлення (DR): співвідношення між кількістю правильно передбачених атак і загальною кількістю атак, також називається коефіцієнтом виявлення (DR) (4.1).

$$TPR = TP / (TP + FN) \quad (4.1)$$

- Рівень хибнопозитивних результатів (FPR): співвідношення між кількістю звичайних випадків, неправильно класифікованих як атаки, та загальною кількістю звичайних випадків (4.2).

$$FPR = FP / (FP + TN) \quad (4.2)$$

- Хибнонегативний показник (FNR): не вдалося визначити аномалію та класифікувати як нормальний трафік (4.3).

$$FNR = FN / (FN + TP) \quad (4.3)$$

- Позитивне прогнозне значення (PPV): ймовірність виявлення вторгнення, якщо IDS видає тривогу (4.4).

$$PPV = TP / (TP + FP) \quad (4.4)$$

- Негативне прогнозоване значення (NPV): ймовірність відсутності вторгнення, коли IDS не видає сигнал тривоги (4.5).

$$NPV = TN / (TN + FN) \quad (4.5)$$

- Коефіцієнт класифікації (CR) або точність: відсоток усіх цих правильно передбачених випадків до всіх випадків, також відомий як точність (4.6).

$$CR = (TP + TN) / (TP + TN + FP + FN) \quad (4.6)$$

- Базова ставка (B): ймовірність що вхідні дані є атакою (4.7).

$$B = (TP + FN) / (TP + TN + FP + FN) \quad (4.7)$$

- Можливість виявлення вторгнень (CID): співвідношення спільної інформації між входом і виходом та ентропією входу (4.8).

$$CID = (H(X) - H(X \mid Y)) / H(X) \quad (4.8)$$

- Ускладнена формула визначення ентропії з використанням PPV і NPV (4.9)

$$h_{x_y} = -b * (1 - fnr) * math.\log(ppv) - b * fnr * math.log(1 - npv) - (1 - b) * (1 - fpr) * math.log(npv) - (1 - b) * fpr * math.log(1 - ppv) \quad (4.9)$$

Далі зробити імпорт набору даних датасету NSL-KDD (рис 4.14). Ці дані будуть включати такі набори:

- train: повний набір навчальних даних
- test: повний набір даних тестування
- test_21: підмножина повного тестового набору даних, у якому беруться лише з'єднання з рівнем складності «21», це «найлегші» з'єднання для класифікації
- test_n21: найскладніший набір даних тестування, з розширеним набором атак, відсутніх у train.

```
train = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTrain+.txt',
                    sep=',', header=None)
test = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTest+.txt',
                   sep=',', header=None)
test_n21 = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTest-21.txt',
                       sep=',', header=None)
```

Рисунок 4.14 - імпорт набору даних датасету NSL-KDD у задані набори

Потім train ділиться на підмножини для кожного макрокласу атаки, а саме NORMAL (нормальний трафік), R2L (remote to user), U2R (user to root), DOS (відмова в обслуговування) і PROBING (зондування) (рис. 4.15).

```
dos_attacks = {'back', 'land', 'neptune', 'pod', 'smurf',
               'teardrop', 'apache2', 'mailbomb',
               'processtable', 'snmpgetattack', 'udpstorm'}
r2l_attacks = {'ftp_write', 'guess_passwd', 'imap',
               'multihop', 'phf', 'spy',
               'warezclient', 'warezmaster', 'snmpguess', 'multihop',
               'named', 'sendmail', 'xlock', 'xsnoop', 'worm'}
u2r_attacks = {'buffer_overflow', 'perl', 'loadmodule',
               'rootkit', 'ps', 'xterm', 'sqlattack', 'httptunnel'}
probing_attacks = {'ipsweep', 'nmap', 'portsweep',
                   'satan', 'mscan', 'saint', 'worm'}
```

Рисунок 4.15 – Поділ імпортованих даних на макрокласи атак

Далі тренуємо і тестуємо test, test_n21 та test_21 (рис. 4.16)

```
rand_int = random.randint(1, len(train) - 1)
x1 = train.iloc[rand_int].values
rand_int = random.randint(1, len(train) - 1)
x2 = train.iloc[rand_int].values
rand_int = random.randint(1, len(train) - 1)
x3 = train.iloc[rand_int].values

s = FG_SOINN(x1, x2, x3, max_edge_age=100, iter_lambda=100)

xs, n_nodes, n_edges, n_del_nodes, n_del_edges = train_phase(model=s,
```

```
Час навчання: 21 хв 29 сек
Вхідні дані оброблено: 125972
Кількість вузлів: 1076
Кількість ребер: 4712
```

Рисунок 4.16 – Тренування з допомогою алгоритму FG-SOINN

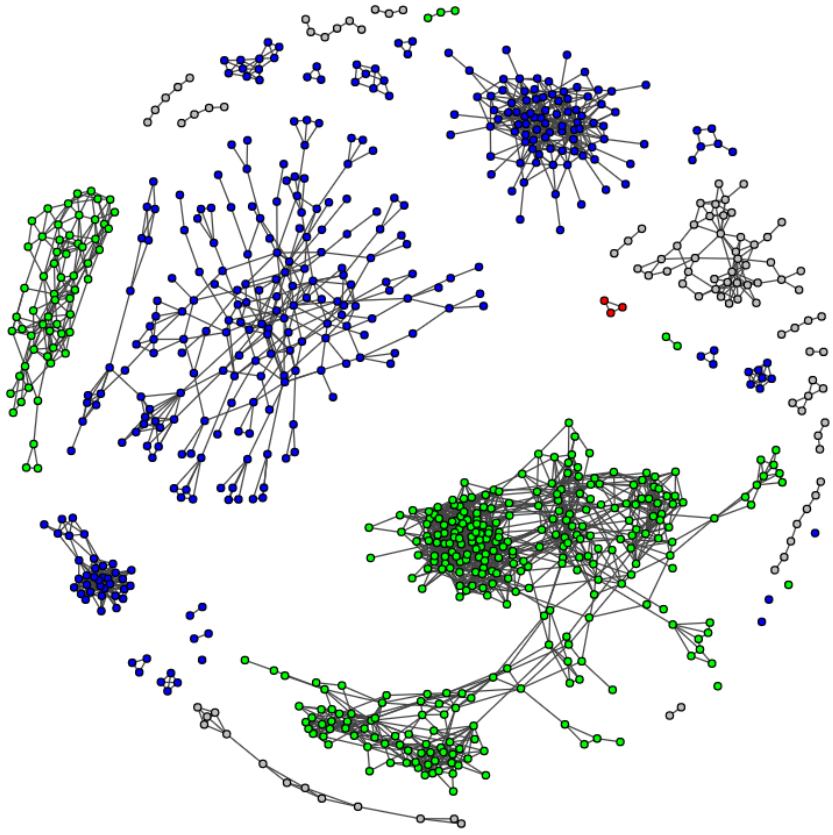


Рисунок 4.17 – Мічений граф для тренованої вибірки

Для наглядності будуємо мічений граф з допомогою igraph для тренованих даних (рис. 4.17) де кольори відповідають типу атаки. Наприклад зелений відповідає нормальному трафіку, синій – атакам DOS, червоний – R2L атакам, чорний – U2R атакам, а сірий – атакам зондування.

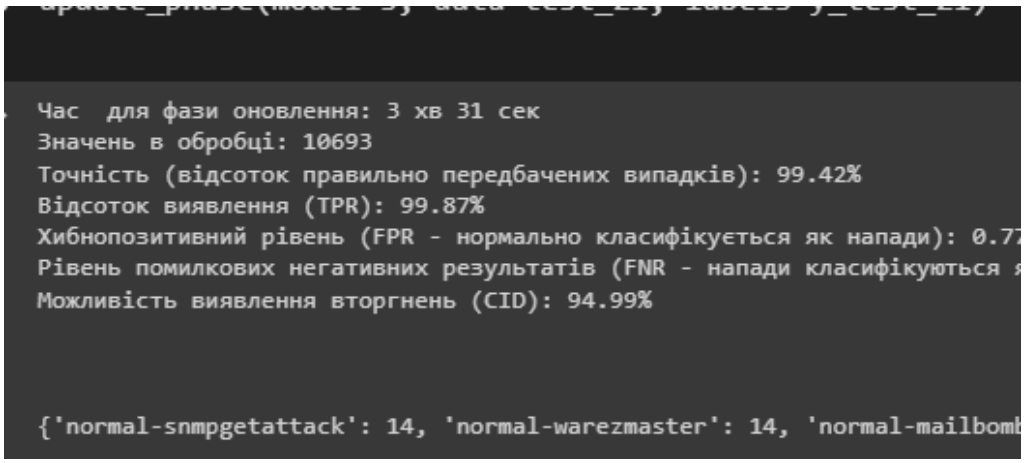


Рисунок 4.18 – Тестування навченої системи

Тестуємо навчену систему (рис. 4.18)

4.4 Експериментальне дослідження

Здійснення ефективності отриманої системи була виконана на наборі даних NSL-KDD, який є вдосконаленою версією відомого набору даних KDD'99. Незважаючи на свій вік, набір даних все ще є де-факто альтернативою для методів і інструментів порівняльного аналізу, які спрямовані на забезпечення ефективних систем виявлення вторгнень. З огляду на його широке використання, що полегшує надання довідкового аналізу, було прийнято цей набір даних для початкового тестування впровадженого методу. Для негативного SOINN було використано n=2, а для позитивного SOINN n=100. Оскільки набір даних включав суміш звичайного трафіку та чотирьох атак, для цього експерименту було

створено п’ять бінарних класів – звичайний клас і чотири класи атак (відмова в обслуговуванні, зондування, R2L і U2R). Створимо вибірку test яка буде включати ці чотири класи атак. Навчаємо систему з допомогою тестових даних test (рис. 4.19). Кількість навчальних даних 22543. Далі запускаємо фазу оновлення, яка буде використовувати невизначені загрози для того, щоб їх «довчити».

```
rand_int = random.randint(1, len(test) - 1)
x1 = test.iloc[rand_int].values
rand_int = random.randint(1, len(test) - 1)
x2 = test.iloc[rand_int].values
rand_int = random.randint(1, len(test) - 1)
x3 = test.iloc[rand_int].values

s = FG_SOINN(x1, x2, x3, max_edge_age=100, iter_lambda=100)

_, _, _, _, _ = train_phase(model=s, data=test, labels=y_test)
```

```
Час навчання: 1 хв 11 сек
Вхідні дані оброблено: 22543
Кількість вузлів: 339
Кількість ребер: 799
```

Рисунок 4.19 – Навчання системи вибіркою test

```
update_phase(model=s, data=train_1, labels=y_train_1)
```

```
Час  для фази оновлення: 1 хв 9 сек
Точність (відсоток правильно передбачених випадків): 96.45%
Відсоток виявлення (TPR): 97.7%
Хибнопозитивний рівень (FPR - нормально класифікується як напади): 4.65%
Рівень помилкових негативних результатів (FNR - напади класифікуються як нормальні): 2.3%
Можливість виявлення вторгнень (CID): 78.31%



{'normal-back': 117, 'back-normal': 39, 'normal-warezclient': 35, 'satan-normal': 27, 'nep
```

Рисунок 4.20 – Фаза першого оновлення

Перша фаза оновлення показала наступні результати (рис. 4.20). Точність 96.44%, відсоток виявлення 97.7, класифікація як напад 4.65%, рівень помилкових результатів 2.3%, можливість виявлення вторгнень 78%.

Для порівняння також варто звернути увагу на восьме оновлення (рис. 4.21) та дванадцяте оновлення (рис. 4.22).

```
update_phase(model=s, data=train_8, labels=y_train_8)

Час  для фази оновлення: 1 хв 20 сек
Точність (відсоток правильно передбачених випадків): 97.5%
Відсоток виявлення (TPR): 97.93%
Хибнопозитивний рівень (FPR - нормально класифікується як напади): 2.88%
Рівень помилкових негативних результатів (FNR - напади класифікуються як нормальні): 2.07%
Можливість виявлення вторгнень (CID): 83.23%



{'normal-warezclient': 54, 'normal-back': 31, 'back-normal': 26, 'normal-satan': 24, 'warez
```

Рисунок 4.21 – Фаза восьмого оновлення

```
[90] update_phase(model=s, data=train_12, labels=y_train_12)

Час  для фази оновлення: 1 хв 19 сек
Точність (відсоток правильно передбачених випадків): 98.23%
Відсоток виявлення (TPR): 98.31%
Хибнопозитивний рівень (FPR - нормально класифікується як напади): 1.83%
Рівень помилкових негативних результатів (FNR - напади класифікуються як нормальні): 1.69%
Можливість виявлення вторгнень (CID): 87.15%



{'normal-back': 28, 'normal-satan': 25, 'normal-warezclient': 24, 'back-normal': 21, 'warezcl
```

Рисунок 4.22 – Фаза дванадцятого оновлення

Якщо порівняти дані з першого, восьмого та дванадцятого оновлення можна побачити явне покращення результатів виявлення атак. В першому невизначені загрози становили 2.3% коли у дванадцятому вони стали 1.69%. Результати всіх дванадцяти оновлень показані в таблиці 3.

Таблиця 3. Результати тестування результатів впровадженої системи на основі FG-SOINN

№	Час, сек	Точність,%	TPR,%	FPR,%	FNR,%	CID,%
1	69	96.45	97.7	4.65	2.3	78.31
2	72	96.73	97.4	4.6	2.26	79.02
3	150	96.83	97.47	3.72	2.53	79.88
4	75	97.66	98.1	2.72	2.49	84.09
5	80	97.67	97.55	2.23	2.45	84.1
6	79	97.7	97.61	2.71	2.39	82.83
7	77	97.67	97.62	1.91	2.81	83.99
8	80	97.5	97.93	2.88	2.07	83.23
9	84	97.52	97.96	2.87	2.24	83.77
10	87	97.92	98.24	2.35	1.76	85.47
11	80	98.06	97.65	1.57	2.15	86.19
12	79	98.23	98.31	1.83	1.69	87.15

Згідно таблиці 3 протягом дванадцяти циклів оновлення рузультат виявлення вторгнень CID постійно збільшувався, а рівень помилкових результатів постійно змінювався, але на дванадцяте оновлення система досягла найкращих результатів.

Слід зазначити, що після початкового навчання для кожного раунду оновлення підмножина перевіряється на навченій FG-SOINN, і лише невдалі прогнози повертаються в систему.

В таблиці 4 проказано ефективність виявлення вторгнень системи виявлення вторгнень які були навчені алгоритмами SOINN та FG-SOINN.

Таблиця 4. Порівняння точності системи IDS навчених алгоритмами SOINN та FG-SOINN

	FG-SOINN	SOINN
1	78.31%	78.4%
2	79.02%	78.46%
3	79.88%	79.25%
4	84.09%	80.41%
5	84.1%	81.12%
6	82.83%	81.4%
7	83.99%	81.41%
8	83.23%	81.72%
9	83.77%	82.11%
10	85.47%	83.6%
11	86.19%	83.9%
12	87.15%	83.92%

Порівняння відбулося з допомогою тестової вибірки test. Дані представляють показник CID (можливість виявлення вторгнень системою ids) у періоді з дванадцяти оновлень. Спочатку з допомогою тестової вибірки test було навчено обидві мережі: мережу на основі алгоритму SOINN та мережу, що використовує FG-SOINN. Потім відбулося тестування на визначення вторгнень з допомогою обидвох систем. Потім на основі зашумлених даних системи почергово донавчалися.

Для візуалізації дані отримані після донавчання були представлені на лінійному графіку, що зображений на рисунку 4.23. Синім кольором представлено відсоткове значення результатів системи яка навчена з допомогою алгоритму FG-SOINN, червоним – SOINN.

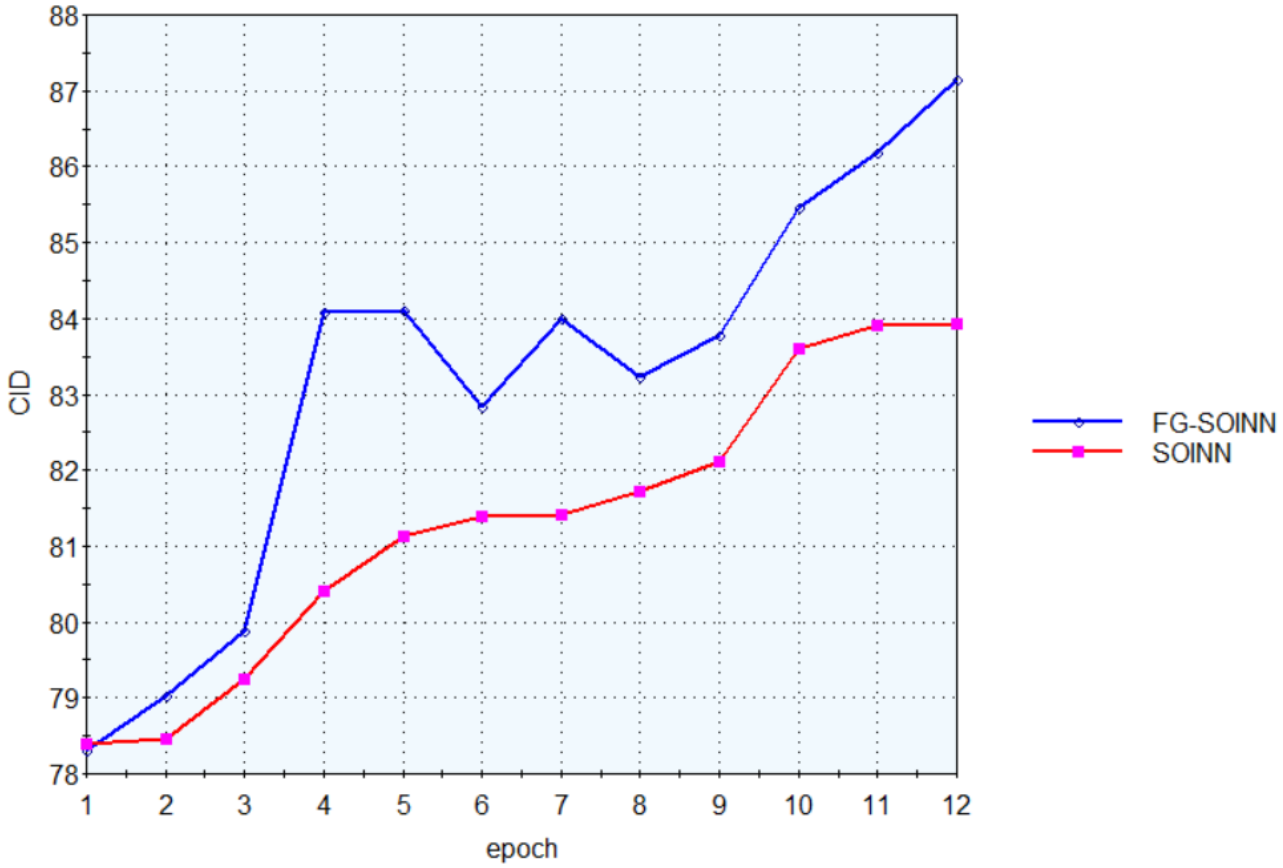


Рисунок 4.23 – Порівняння систем на основі алгоритмів SOINN та GF-SOINN

Дані показники показують, що точність навчання алгоритму GF-SOINN більша ніж SOINN. Якість виявлення вторгнень системою IDS покращується з кожною новою фазою, що підтверджує твердження про ефективність запропонованої структури.

Висновки до четвертого розділу

В ході роботи над четвертим розділом було реалізовано самоорганізовану систему виявлення зловмисної активності в комп’ютерних мережах.

Незважаючи на те, що час витрачений на навчання системи значно збільшується зі збільшенням кількості вхідних даних оновлення, режими

оновлення та робочий режим може працювати паралельно або з можливістю для перемикання, коли робоча фаза неактивна (тобто відсутні вхідні дані для виявлення). Таким чином вона набуває більше можливостей шляхом вивчення невдалих класифікацій або нових вхідних даних. Саме такі здібності змушують її вважатися самоорганізованою.

Така система здатна прогнозувати зловмисну активність та одночасно навчатися на невдалих попередніх спробах, що робить її більш ефективною перед штучними та реальними даними.

ВИСНОВКИ

В ході розробки даної магістерської роботи було досліджено предметну область, аналізовано теоретичну інформацію, різноманітні засоби і способи захисту мережевого трафіку. Було спроектовано та реалізовано розподілену самоорганізовану систему прогнозування зловмисної активності, а саме системи виявлення вторгнень на основі штучного інтелекту та модифікованого модуля самоорганізованої інкрементної нейронної мережі. Такий модуль повинен використовувати структуру техніки «сміттєвого забуття» на основі концепцій часу простою, надійності та корисності. Початок роботи пов’язаний з аналізом систем прогнозування зловмисної активності та дослідження впливу на неї. Також в роботі був впроваджений метод реалізації самоорганізованих систем прогнозування зловмисної активності на основі інкрементного навчання. Реалізується він з допомогою системи виявлення вторгнень яку навчають з допомогою нейронних мереж. Запропонована розподілена структура та її результати показують те, що така пропозиція може адаптуватися до мережевих даних як для звичайних категорій, так і для категорій атак. Така система використовує значно менше ресурсів, є швидшою і має кращу швидкість виявлення, ніж метод навчання з вчителем. Забезпечення системи невдалими прогнозами, ми не тільки досягаємо поступового навчання з багатообіцяючою точністю, але й ефективну структуру, тобто замість того, щоб подавати їй повний набір даних, FG-SOINN зберігає лише частину набору даних, роблячи дану структуру дуже хорошим кандидатом для систем масштабування.

Розроблене ПЗ було реалізоване з допомогою середовища Jupiter Notebook та мови програмування Python. Я вважаю, що поставлена задача виконана в повному обсязі. В разі потреби, цю систему та рішення можна доповнити, розширити або удосконалити.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Madarie R. Hackers' Motivations: Testing Schwartz's Theory of Motivational Types of Values in a Sample of Hackers //International Journal of Cyber Criminology. – 2017. – Т. 11. – №. 1.

2. Al-Qurishi M. et al. Leveraging analysis of user behavior to identify malicious activities in large-scale social networks //IEEE Transactions on Industrial Informatics. – 2017. – Т. 14. – №. 2. – С. 799-813.

3. Fehr C., LiCalzi C., Oates T. Computer crimes //Am. Crim. L. Rev. – 2016. – Т. 53. – С. 977.

4. Brdiczka, O., J. Liu, B. Price, J. Shen. Proactive Insider Threat Detection Through GraphLearning & Psychological Context. Palo Alto Research Centre.

5. Lee T. F. Provably secure anonymous single-sign-on authentication mechanisms using extended Chebyshev chaotic maps for distributed computer networks //IEEE Systems Journal. – 2015. – Т. 12. – №. 2. – С. 1499-1505.

6. Guo B. Why hackers become crackers–An analysis of conflicts faced by hackers //Public Administration Research. – 2016. – Т. 5. – №. 1. – С. 29.

7. Passeri, P. Cyber Attack Statistics. Hackmageddon.com, 2014.

8. Tang Y., Elhoseny M. Computer network security evaluation simulation model based on neural network //Journal of Intelligent & Fuzzy Systems. – 2019. – Т. 37. – №. 3. – С. 3197-3204.

9. Rezende E. et al. Malicious software classification using transfer learning of resnet-50 deep neural network //2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). – IEEE, 2017. – С. 1011-1014.

10. Rezende E. et al. Malicious software classification using VGG16 deep neural network’s bottleneck features //Information technology-new generations. – Springer, Cham, 2018. – С. 51-59.

11. Vinod P. et al. Detecting malicious files using non-signature-based methods //International Journal of Information and computer security. – 2014. – Т. 6. – №. 3. – С. 199-240.

12. Attaluri, S., McGhee, S. and Stamp, M. (2009) 'Profile hidden Markov models and metamorphicvirus detection', Journal in Computer Virology, Vol. 5, No. 2, pp.151–169.

13. Baysa, D., Low, R.M. and Stamp, M. (2013) 'Structural entropy and metamorphicmalware', Journal of Computer Virology and Hacking Techniques, Vol. 9, No. 4, pp.1–14,Springer-Verlag.

14. Francesco Mercaldo, Vittoria Nardone, and Antonella Santone. Ransomware inside out. In 2016 11th International Conference on Availability, Reliability and Security (ARES) , pages 628–637. IEEE, 2016.

15. Steve Mansfield-Devine. Ransomware: taking businesses hostage. Network Security , 2016(10):8–17,2016.

16. Cath Everett. Ransomware: to pay or not to pay? Computer Fraud & Security , 2016(4):8–12, 2016.

17. Gu X. et al. Deep API learning //Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. – 2016. – С. 631-642.

18. Pavlenko I. et al. Application of artificial neural network for identification of bearing stiffness characteristics in rotor dynamics analysis //Design, simulation, manufacturing: the innovation exchange. – Springer, Cham, 2018. – С. 325-335.

19. Brunton S. L., Kutz J. N. Data-driven science and engineering: Machine learning, dynamical systems, and control. – Cambridge University Press, 2022.

20. Moskovitch R. Multivariate temporal data analysis-a review //Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery. – 2022. – Т. 12. – №. 1. – С. e1430.

21. Wang Y. et al. Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems //Concurrency and Computation: Practice and Experience. – 2019. – Т. 31. – №. 19. – С. e5101.

22. Азарова А. О., Білий Р. О. Аналіз базових концепцій розподіленої корпоративної мережі : дис. – ВНТУ, 2021.

23. Ge X. et al. Distributed event-triggered estimation over sensor networks: A survey //IEEE transactions on cybernetics. – 2019. – Т. 50. – №. 3. – С. 1306-1320.

24. Васильківський М. В., Скощук В. К. Захист інформації в мультисервісних мережах зв'язку : дис. – ВНТУ, 2020.

25. Любінецький Д. В. Інтелектуальна система аналізу трафіку в складі комплексної системи захисту. – 2021.

26. Gümüşbaş D. et al. A comprehensive survey of databases and deep learning methods for cybersecurity and intrusion detection systems //IEEE Systems Journal. – 2020. – Т. 15. – №. 2. – С. 1717-1731.

27. Mhawi D. N., Aldallal A., Hassan S. Advanced feature-selection-based hybrid ensemble learning algorithms for network intrusion detection systems //Symmetry. – 2022. – Т. 14. – №. 7. – С. 1461.

28. Wu Y., Wei D., Feng J. Network attacks detection methods based on deep learning techniques: a survey //Security and Communication Networks. – 2020. – Т. 2020.

29. Manavi M. T. Defense mechanisms against distributed denial of service attacks: A survey //Computers & Electrical Engineering. – 2018. – Т. 72. – С. 26-38.

30. Colom J. F. et al. Scheduling framework for distributed intrusion detection systems over heterogeneous network architectures //Journal of Network and Computer Applications. – 2018. – Т. 108. – С. 76-86.

31. Hajiheidari S. et al. Intrusion detection systems in the Internet of things: A comprehensive investigation //Computer Networks. – 2019. – Т. 160. – С. 165-191.

32. Özalp M., Karakuzu C., Zengin A. Distributed intrusion detection systems: A survey //Academic Perspective Procedia. – 2019. – Т. 2. – №. 3. – С. 400-407.

33. Din M. S. U., Rehman M. A. U., Kim B. S. CIDF-WSN: A Collaborative interest and data forwarding strategy for named data wireless sensor networks //Sensors. – 2021. – Т. 21. – №. 15. – С. 5174.

34. Tang B. et al. Recent advances of deep learning in bioinformatics and computational biology //Frontiers in genetics. – 2019. – Т. 10. – С. 214.

35. Schrage B. et al. Association between use of primary-prevention implantable cardioverter-defibrillators and mortality in patients with heart failure: a prospective propensity score–matched analysis from the Swedish Heart Failure Registry //Circulation. – 2019. – Т. 140. – №. 19. – С. 1530-1539.

36. Chaabouni N. et al. Network intrusion detection for IoT security based on learning techniques //IEEE Communications Surveys & Tutorials. – 2019. – Т. 21. – №. 3. – С. 2671-2701.

37. Герасименко С. О. Система безпеки розподіленого зберігання даних : дис. – КПІ ім. Ігоря Сікорського, 2022.

38. Савенко Б. О. Самоорганізована розподілена система виявлення аномалій в комп'ютерних системах на основі методу головних компонент. – 2021.

39. Лісова В. П. Використання механізмів серверних операційних систем щодо пошуку інсайдерів у корпоративних мережах. – 2020.

40. Qaddoura R. et al. A multi-layer classification approach for intrusion detection in iot networks based on deep learning //Sensors. – 2021. – Т. 21. – №. 9. – С. 2987.

41. Chawla A. et al. Host based intrusion detection system with combined CNN/RNN model //Joint European conference on machine learning and knowledge discovery in databases. – Springer, Cham, 2018. – С. 149-158.

42. Дорогий Я. Ю., Свириденко О. А. Тактичний штучний інтелект з використанням нейронної мережі для стратегії в реальному часі //Проблеми інформатизації та управління. – Т. 2. – №. 66. – С. 27-33.

43. Мєшков М. О. ШТУЧНИЙ ІНТЕЛЕКТ ЯК БАЗА МОДЕЛЮВАННЯ НЕЙРОННИХ МЕРЕЖ //RESEARCH OF DIFFERENT DIRECTIONS OF DEVELOPMENT OF PHILOLOGICAL SCIENCES IN UKRAINE AND EU. – 2019. – С. 116.

44. Brundage M. et al. The malicious use of artificial intelligence: Forecasting, prevention, and mitigation //arXiv preprint arXiv:1802.07228. – 2018.

45. Madan R., Mangipudi P. S. Predicting computer network traffic: a time series forecasting approach using DWT, ARIMA and RNN //2018 Eleventh International Conference on Contemporary Computing (IC3). – IEEE, 2018. – С. 1-5.

46. Jiang W., Luo J. Graph neural network for traffic forecasting: A survey //Expert Systems with Applications. – 2022. – С. 117921.

47. Ertam F. An efficient hybrid deep learning approach for internet security //Physica A: Statistical Mechanics and Its Applications. – 2019. – Т. 535. – С. 122492.

48. Vafeiadis T., Papanikolaou A., Ilioudis C. & CharchalakisS. (2012). Real-time network data analysis using timeseries models, Elsevier Simulation Modelling Practice andTheory, 29, 173-180.http://doi.org/10.1016/j.simpat.2012.07.002

49. Katris C. & Daskalaki S. (2015). Comparing forecastingapproaches for Internet traffic, Elsevier Expert Systemswith Applications, 42(21), 8172-8183.http://doi.org/10.1016/j.eswa.2015.06.029

50. ErtamF. et al. Network traffic classification via kernel based extreme learning machine Int. J. Intell. Syst. Appl. Eng. (2016)

51. RoughanM. et al. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification

52. Sun G. et al. Internet traffic classification based on incremental support vector machines //Mobile Networks and Applications. – 2018. – Т. 23. – №. 4. – С. 789-796.

53. AuldT. et al. Bayesian neural networks for internet traffic classification IEEE Trans. Neural Netw. (2017)

54. SunR. et al.Traffic classification using probabilistic neural networks

55. Sun G. et al. Internet traffic classification based on incremental support vector machines //Mobile Networks and Applications. – 2018. – Т. 23. – №. 4. – С. 789-796.

56. Dong S. Multi class SVM algorithm with active learning for network traffic classification //Expert Systems with Applications. – 2021. – Т. 176. – С. 114885.

57. WangR. et al. Solving the app-level classification problem of P2P traffic via optimized support vector machines

58. Wang P. et al. Identifying peer-to-peer botnets through periodicity behavior analysis //2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). – IEEE, 2018. – С. 283-288.

59. Alkhalaf S. A control-driven autonomous authentication scheme for peer-to-peer control systems assisted industrial Internet of things //Soft Computing. – 2021. – Т. 25. – №. 18. – С. 12175-12189.

60. Kozak Y. Прогнозування трафіку корпоративної мережі із застосуванням штучних нейронних мереж //COMPUTER-INTEGRATED TECHNOLOGIES: EDUCATION, SCIENCE, PRODUCTION. – 2022. – №. 47. – С. 98-104.

61. Han J. et al. Polycyclic aromatic hydrocarbon (PAHs) geographical distribution in China and their source, risk assessment analysis //Environmental Pollution. – 2019. – Т. 251. – С. 312-327.

62. Nguyen H. et al. A comparative study of artificial neural networks in predicting blast-induced air-blast overpressure at Deo Nai open-pit coal mine, Vietnam //Neural Computing and Applications. – 2020. – Т. 32. – №. 8. – С. 3939-3955.

63. Cortez et al., 2012; Hallas and Dorfner, 1998; Dean et al., 1995; Feng and Shu, 2005

64. Dias K. L. et al. An innovative approach for real-time network traffic classification //Computer Networks. – 2019. – Т. 158. – С. 143-157.

65. Nie L. et al. Network traffic prediction in industrial Internet of Things backbone networks: A multitask learning mechanism //IEEE Transactions on Industrial Informatics. – 2021. – Т. 17. – №. 10. – С. 7123-7132.

66. Cabaneros S. M., Calautit J. K., Hughes B. R. A review of artificial neural network models for ambient air pollution prediction //Environmental Modelling & Software. – 2019. – Т. 119. – С. 285-304.

67. Omari M. A. et al. Modeling of the viscoelastic properties of thermoset vinyl ester nanocomposite using artificial neural network //International Journal of Engineering Science. – 2020. – Т. 150. – С. 103242.

68. Lopez-Cortes A. et al. In silico analyses of immune system protein interactome network, single-cell RNA sequencing of human tissues, and artificial neural networks reveal potential therapeutic targets for drug repurposing against COVID-19 //Frontiers in Pharmacology. – 2021. – Т. 12. – С. 598925.

69. Toneva M. et al. An empirical study of example forgetting during deep neural network learning //arXiv preprint arXiv:1812.05159. – 2018.

70. Hinton G. How to represent part-whole hierarchies in a neural network //arXiv preprint arXiv:2102.12627. – 2021.

71. Hooker S. et al. A benchmark for interpretability methods in deep neural networks //Advances in neural information processing systems. – 2019. – Т. 32.

72. Guo H. et al. Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach //Resources Policy. – 2021. – Т. 74. – С. 101474.

73. Keriven N., Peyré G. Universal invariant and equivariant graph neural networks //Advances in Neural Information Processing Systems. – 2019. – Т. 32.

74. Davenport T. H. The AI advantage: How to put the artificial intelligence revolution to work. – mit Press, 2018.

75. Hoang A. T. et al. A review on application of artificial neural network (ANN) for performance and emission characteristics of diesel engine fueled with biodiesel-based fuels //Sustainable Energy Technologies and Assessments. – 2021. – Т. 47. – С. 101416.

76. Ghorbanian, M. Ahmadi, R. Soltani, “Design predictive tool & optimization of journal bearing using neural network model & multi objective genetic algorithm”, Scientia Iranica B (2011), Volume 18, No.5 Page no.-1095-1105

77. Li C. et al. Efficient and self-adaptive in-situ learning in multilayer memristor neural networks //Nature communications. – 2018. – Т. 9. – №. 1. – С. 1-8.

78. Raghavan G., Lin C., Thomson M. Self-organization of multi-layer spiking neural networks //arXiv preprint arXiv:2006.06902. – 2020.

79. Lin P. et al. Three-dimensional memristor circuits as complex neural networks //Nature Electronics. – 2020. – Т. 3. – №. 4. – С. 225-232.

80. Lin H. et al. Data-based fault tolerant control for affine nonlinear systems through particle swarm optimized neural networks //IEEE/CAA Journal of Automatica Sinica. – 2020. – Т. 7. – №. 4. – С. 954-964.

81. S. Furao and O. Hasegawa, “An incremental network for on-line unsupervised classification and topology learning,” Neural Networks, vol. 19, no. 1, pp. 90– 106, 2006.

82. Каштальян А.С., Любінецький Д.В., Розподілена самоорганізована система прогнозування зловмисної активності в комп’ютерних мережах.

Вимірювальна та обчислювальна техніка в технологічних процесах. Технічні науки. 2022. № 4. С. 22-26.

83. Любінецький Д.В., Система захисту комп’ютерної мережі від інсайдеських атак / О.С. Савенко, І.В. Муляр, Д.В. Любінецький // Збірник наукових праць Військового інституту Київського національного університету імені Тараса Шевченка. – К.: ВІКНУ, 2022. – Вип. №75. – С. 56.

Додаток А
(Обов’язковий)
Лістинг програми

FG-SOINN.py

```
import numpy as np
import igraph as ig
from numpy.linalg import norm
import math
import random
from collections import Counter
NOISE_LABEL = 'noise'
class FG _SOINN(object):
    def __init__(self, x1, x2, x3, max_edge_age, iter_lambda=100, pull_factor=100):
      self.t = 3
      self.max_edge_age = max_edge_age
      self.iter_lambda = iter_lambda
      self.pull_factor = pull_factor
      self.n_del_edges = 0
      self.n_del_nodes = 0
      self.network = ig.Graph()
      self.network.add_vertices(3)
      self.network.vs[0]['w'] = x1
      self.network.vs[1]['w'] = x2
      self.network.vs[2]['w'] = x3
      self.network.vs[0]['wt'] = 1
      self.network.vs[1]['wt'] = 1
      self.network.vs[2]['wt'] = 1
      self.network.vs[0]['it'] = 1
      self.network.vs[1]['it'] = 1
      self.network.vs[2]['it'] = 1
      self.network.vs[0]['cl'] = [NOISE_LABEL]
      self.network.vs[1]['cl'] = [NOISE_LABEL]
      self.network.vs[2]['cl'] = [NOISE_LABEL]
      self.network.vs[0]['c'] = NOISE_LABEL
      self.network.vs[1]['c'] = NOISE_LABEL
```

```
    self.network.vs[2]['c'] = NOISE_LABEL
def _distance(self, a, b):
    f = 0.5
    diff = abs(a - b) ** f
    sum = np.sum(diff)
    distance = math.pow(sum, 1/f)
    return distance
def _get_n1_n2(self, x):
    distances = dict()
    for n in self.network.vs:
        distances[n] = self._distance(x, n['w'])
    n1 = min(distances, key=distances.get)
    del distances[n1]
    n2 = min(distances, key=distances.get)
    return n1, n2
def _similarity_threshold(self, x, node):
    distances = []
      if node.degree() == 0:
       for n in self.network.vs:
           if n.index != node.index:
               distances.append(self._distance(node['w'], n['w']))
       d = min(distances)
    else: for n in node.neighbors():
           distances.append(self._distance(node['w'], n['w']))
       d = max(distances)
           if d != 0:
       d += d * (1 - 1 / node['wt'])
    node['st'] = d
def _add_node(self, weights, y):
    new_node = self.network.add_vertex()
    new_node['w'] = weights
    new_node['wt'] = 1
    new_node['it'] = 1
    new_node['cl'] = [y]
    new_node['c'] = y
def _merge_nodes(self, n1, x):
```

```
    n1['w'] = n1['w'] + ((x - n1['w']) / n1['wt'])
    for n in n1.neighbors():
        pulled_weight = self.pull_factor * n['wt']
        n['w'] = n['w'] + ((x - n['w']) / pulled_weight)
    n1['it'] = 1
def _linking(self, n1, n2):
    n_edges = self.network.ecount()
    if n_edges == 0 or not self.network.are_connected(n1.index, n2.index):
        edge = self.network.add_edge(source=n1.index, target=n2.index)
        edge['it'] = 1
    else: self.network.es[self.network.get_eid(n1.index, n2.index)]['it'] = 0
    for e in self.network.vs[n1.index].incident():
        e['it'] += 1
def _edge_deletion(self):
        for e in self.network.es:
        source = self.network.vs[e.source]
        target = self.network.vs[e.target]
        if e['it'] > self.max_edge_age or source['c'] != target['c']:
            self.network.delete_edges(e.index)
            self.n_del_edges += 1
def _nodes_deletion(self):
    max_u = 0.0
    us = []
    for n in self.network.vs:
        n['u'] = n['wt'] / n['it']
        us.append(n['u'])
        if n['u'] > max_u:
            max_u = n['u']
    u_mean = np.mean(us)
    for n in self.network.vs:
        if n.degree() == 0 and n['u'] < u_mean:
            prob_survival = n['u'] / max_u
            prob_deletion = 1 - prob_survival
            if prob_deletion > prob_survival:
                self.network.delete_vertices(n.index)
                self.n_del_nodes += 1
```

```
    for n in self.network.vs:
        n['it'] += 1
def _group(self):
            for n in self.network.vs:
        occurence_count = Counter(n['cl'])
        n['c'] = occurence_count.most_common(1)[0][0]
        n['cl'] = [n['c']]
def input_signal(self, x, y=None, learning=True):
    n1, n2 = self._get_n1_n2(x)
    if learning:
        if y is None:
            y = NOISE_LABEL
        self.t += 1
        n_nodes = self.network.vcount()
        n_edges = self.network.ecount()

        prediction = n1['c']
        self._similarity_threshold(x, n1)
        self._similarity_threshold(x, n2)
        d1 = self._distance(x, n1['w'])
        d2 = self._distance(x, n2['w'])
        if d1 >= n1['st'] or d2 >= n2['st']:
            self._add_node(x, y)
        else: n1['wt'] += 1
            if y != NOISE_LABEL:
                n1['cl'].append(y)
            self._merge_nodes(n1, x)
            self._linking(n1, n2)
        if self.t % self.iter_lambda == 0:
            if n_nodes > 3:
                self._nodes_deletion()
            self._group()
            if n_edges > 3:
                self._edge_deletion()
    else:  prediction = n1['c']
    confidence = 0
```

```
        if n1 in self.network.vs:
            self._similarity_threshold(x, n1)
            if n1['st'] != None and n1['st'] != 0:
                confidence = 1 - self._distance(
                    x, n1['w']) / n1['st']
        return prediction, confidence
```

Self_organized_system.py

```
import igraph as ig
import numpy as np
from numpy.linalg import norm
import random
import pandas as pd
import scipy.io
import time
import math
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from IPython.display import display, clear_output
from operator import itemgetter
from sklearn.decomposition import PCA
from SOINN import GF_SOINN
def train_phase(model, data, labels):
    xs = []
    n_nodes = []
    n_edges = []
    n_del_nodes = []
    n_del_edges = []
    start_time = time.time()
    for index, row in data.iterrows():
        model.input_signal(x=row.values, y=labels[index], learning=True)
        percent_completed = round((index / len(data))*100, 2)
        if index % 100 == 0 or index == len(data):
            clear_output(wait=True)
            print(f'Processing input {index}: {percent_completed}% completed')
            n_nodes.append(model.network.vcount())
```

```
            n_edges.append(model.network.ecount())
            n_del_nodes.append(model.n_del_nodes)
            n_del_edges.append(model.n_del_edges)
            xs.append(index)
    finish_time = round(time.time() - start_time)
    mins = round(finish_time / 60)
    secs = finish_time % 60
    clear_output(wait=True)
    print(f'Training time: {mins} min {secs} sec')
    print(f'Inputs processed: {index}')
    print(f'Number of nodes: {model.network.vcount()}')
    print(f'Number of edges: {model.network.ecount()}')
    return xs, n_nodes, n_edges, n_del_nodes, n_del_edges
def update_phase(model, data, labels, show_stats=True):
    show_stats : boolean (default: True)
    predicted = []
    tp = 0     fn = 0     fp = 0     tn = 0     tpr = 1     fpr = 1     fnr = 1     ppv = 1
    npv = 1     accuracy = 1     b = 1     cid = 1
    start_time = time.time()
    for index, row in data.iterrows():
        x = row.values
        y = labels[index]
        yp, _ = model.input_signal(x=x, learning=False)
        predicted.append(yp)
        if y != yp:
            model.input_signal(x=x, y=y, learning=True)
        if yp != 'normal' and y != 'normal':            tp += 1
        if yp != 'normal' and y == 'normal':            fp += 1
        if yp == 'normal' and y == 'normal':            tn += 1
        if yp == 'normal' and y != 'normal':            fn += 1
        percent_completed = round((index / len(data))*100, 2)
        if index % 100 == 0 or index == len(data):
            clear_output(wait=True)
            print(f'Predictions completed: {percent_completed}%')
                clear_output(wait=True)
    if show_stats:
```

```
    finish_time = round(time.time() - start_time)
    mins = round(finish_time / 60)
    secs = finish_time % 60
    if (tp + fn) > 0:
       tpr = tp / (tp + fn)
    if (fp + tn) > 0:
       fpr = fp / (fp + tn)
    if (fn + tp) > 0:
       fnr = fn / (fn + tp)
    if (tp + fp) > 0:
       ppv = tp / (tp + fp)
    if (tn + fn) > 0:
       npv = tn / (tn + fn)
    if (tp + tn + fp + fn) > 0:
       accuracy = (tp + tn) / (tp + tn + fp + fn)
       b = (tp + fn) / (tp + tn + fp + fn)
    if b > 0 and (1-b) > 0:
       h_x = -b * math.log(b) - (1-b) * math.log(1-b)
    else:          h_x = 0
    if (b * (1 - fnr) + (1 - b) * fpr) != 0 and (b * (1 - fnr) / (b * (1 - fnr) + (1 - b) * fpr)):
       h_x_y = -b * (1 - fnr) * math.log(b * (1 - fnr) / (b * (1 - fnr) + (1 - b) * fpr))
    else:          h_x_y = 0
    if (b * fnr + (1 - b) * (1-fpr)) != 0 and (b * fnr / (b * fnr + (1 - b) * (1-fpr))) > 0:
       h_x_y -= b * fnr * math.log( b * fnr / (b * fnr + (1 - b) * (1-fpr)) )
    else:          h_x_y -= 0
    if ((1 - b) * (1 - fpr) + b * fnr) != 0 and ((1 - b) * (1 - fpr) / ((1 - b) * (1 - fpr) + b *
fnr)) > 0:
       h_x_y -= (1 - b) * (1 - fpr) * math.log( (1 - b) * (1 - fpr) / ((1 - b) * (1 - fpr) + b
* fnr) )
    else:          h_x_y -= 0       if ((1 - b) * fpr + b * (1 - fnr)) != 0 and ((1 - b) * fpr /
((1 - b) * fpr + b * (1 - fnr))) > 0:
       h_x_y -= (1 - b) * fpr * math.log( (1 - b) * fpr / ((1 - b) * fpr + b * (1 - fnr)) )
    else:          h_x_y -= 0
    if h_x == 0 or h_x == 1:
       cid = 1       else:          cid = (h_x - h_x_y) / h_x
    clear_output(wait=True)
```

```
        print(f'Час  для фази оновлення: {mins} хв {secs} сек')
        print(f'Значень в обробці: {index}')
        print(f'Точність (відсоток правильно передбачених випадків): {round(accuracy * 100, 2)}%')
        print(f'Відсоток виявлення (TPR): {round(tpr * 100, 2)}%')
        print(f'Хибнопозитивний рівень (FPR - нормально класифікується як напади): {round(fpr * 100, 2)}%')
        print(f'Рівень помилкових негативних результатів (FNR - напади класифікуються як нормальні): {round(fnr * 100, 2)}%')
        print(f'Можливість виявлення вторгнень (CID): {round(cid * 100, 2)}%')
        errors_freq = dict()
        for i in range(len(labels)):
            if labels[i] != predicted[i]:
                errors_freq[f'{labels[i]}-{predicted[i]}'] = 0
        for i in range(len(labels)):
            if labels[i] != predicted[i]:
                errors_freq[f'{labels[i]}-{predicted[i]}'] += 1
        errors_freq = dict(sorted(errors_freq.items(), key=itemgetter(1), reverse=True))
        print(errors_freq)
def live_phase(model, connection, teach=False, label=None):
        if teach:                if label is None:                yp, confidence = model.input_signal(x=connection, learning=True)
        else:                yp, confidence = model.input_signal(x=connection, y=label, learning=True)
    else:        yp, confidence = model.input_signal(x=connection, learning=False)
    return yp, confidence
train = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTrain+.txt', sep=',', header=None)
test = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTest+.txt', sep=',', header=None)
test_n21 = pd.read_csv('/content/drive/MyDrive/NSL-KDD/KDDTest-21.txt', sep=',', header=None)
train.drop([0], axis=1, inplace=True)
test.drop([0], axis=1, inplace=True)
test_n21.drop([0], axis=1, inplace=True)
train = train.sample(frac=1).reset_index(drop=True)
```

```
train = pd.get_dummies(train, columns=[1, 2, 3])
test = pd.get_dummies(test, columns=[1, 2, 3])
test_n21 = pd.get_dummies(test_n21, columns=[1, 2, 3])
extra_columns = []
for i in train.columns.values:
    if i not in test.columns.values:
        extra_columns.append(i)
for col in extra_columns:
    test[col] = 0
extra_columns = []
for i in train.columns.values:
    if i not in test_n21.columns.values:
        extra_columns.append(i)
for col in extra_columns:
    test_n21[col] = 0
train.columns.astype('string')
test.columns.astype('string')
test_n21.columns.astype('string')
cols = train.columns.tolist()
cols.sort()
train = train[cols]
test = test[cols]
test_n21 = test_n21[cols]
test_21 = test[test['42'] == 21]
test_21.reset_index(inplace=True)
del test_21['42']
del test_21['index']
del train['42']
del test['42']
del test_n21['42']
dos_attacks = {'back', 'land', 'neptune', 'pod', 'smurf', 'teardrop', 'apache2', 'mailbomb',
'processtable', 'snmpgetattack', 'udpstorm'}
r2l_attacks = {'ftp_write', 'guess_passwd', 'imap', 'multihop', 'phf', 'spy', 'warezclient',
'warezmaster', 'snmpguess', 'multihop', 'named', 'sendmail', 'xlock', 'xsnoop', 'worm'}
u2r_attacks = {'buffer_overflow', 'perl', 'loadmodule', 'rootkit', 'ps', 'xterm', 'sqlattack',
'httptunnel'}
```

```
probing_attacks = {'ipsweep', 'nmap', 'portsweep', 'satan', 'mscan', 'saint', 'worm'}
train_normal = train[train['41'] == 'normal']
train_dos = train[train['41'].isin(dos_attacks)]
train_r2l = train[train['41'].isin(r2l_attacks)]
train_u2r = train[train['41'].isin(u2r_attacks)]
train_probing = train[train['41'].isin(probing_attacks)]
test_normal = test[test['41'] == 'normal']
test_dos = test[test['41'].isin(dos_attacks)]
test_r2l = test[test['41'].isin(r2l_attacks)]
test_u2r = test[test['41'].isin(u2r_attacks)]
test_probing = test[test['41'].isin(probing_attacks)]
y_train = train['41'].values
del train['41']
y_test = test['41'].values
del test['41']
y_test_21 = test_21['41'].values
del test_21['41']
y_test_n21 = test_n21['41'].values
del test_n21['41']
y_train_normal = train_normal['41'].values
del train_normal['41']
y_train_dos = train_dos['41'].values
del train_dos['41']
y_train_r2l = train_r2l['41'].values
del train_r2l['41']
y_train_u2r = train_u2r['41'].values
del train_u2r['41']
y_train_probing = train_probing['41'].values
del train_probing['41']
y_test_normal = test_normal['41'].values
del test_normal['41']
y_test_dos = test_dos['41'].values
del test_dos['41']
y_test_r2l = test_r2l['41'].values
del test_r2l['41']
y_test_u2r = test_u2r['41'].values
```

```
del test_u2r['41'] y_test_probing = test_probing['41'].values
del test_probing['41']
mms = MinMaxScaler()
train[train.columns] = mms.fit_transform(train)
test[test.columns] = mms.transform(test)
test_n21[test_n21.columns] = mms.transform(test_n21)
test_21[test_21.columns] = mms.transform(test_21)
train_normal[train_normal.columns] = mms.transform(train_normal)
train_dos[train_dos.columns] = mms.transform(train_dos)
train_r2l[train_r2l.columns] = mms.transform(train_r2l)
train_u2r[train_u2r.columns] = mms.transform(train_u2r)
train_probing[train_probing.columns] = mms.transform(train_probing)
test_normal[test_normal.columns] = mms.transform(test_normal)
test_dos[test_dos.columns] = mms.transform(test_dos)
test_r2l[test_r2l.columns] = mms.transform(test_r2l)
test_u2r[test_u2r.columns] = mms.transform(test_u2r)
test_probing[test_probing.columns] = mms.transform(test_probing)
train_1 = train[:10000]
train_2 = train[10000:20000]
train_3 = train[20000:30000]
train_4 = train[30000:40000]
train_5 = train[40000:50000]
train_6 = train[50000:60000]
train_7 = train[60000:70000]
train_8 = train[70000:80000]
train_9 = train[80000:90000]
train_10 = train[90000:100000]
train_11 = train[100000:110000]
train_12 = train[110000:120000]
train_13 = train[120000:]
y_train_1 = y_train[:10000]
y_train_2 = y_train[10000:20000]
y_train_3 = y_train[20000:30000]
y_train_4 = y_train[30000:40000]
y_train_5 = y_train[40000:50000]
y_train_6 = y_train[50000:60000]
```

```
y_train_7 = y_train[60000:70000]
y_train_8 = y_train[70000:80000]
y_train_9 = y_train[80000:90000]
y_train_10 = y_train[90000:100000]
y_train_11 = y_train[100000:110000]
y_train_12 = y_train[110000:120000]
y_train_13 = y_train[120000:]
train_2.reset_index(inplace=True)
train_3.reset_index(inplace=True)
train_4.reset_index(inplace=True)
train_5.reset_index(inplace=True)
train_6.reset_index(inplace=True)
train_7.reset_index(inplace=True)
train_8.reset_index(inplace=True)
train_9.reset_index(inplace=True)
train_10.reset_index(inplace=True)
train_11.reset_index(inplace=True)
train_12.reset_index(inplace=True)
train_13.reset_index(inplace=True)
del train_2['index'] del train_3['index']
del train_4['index']
del train_5['index']
del train_6['index']
del train_7['index']
del train_8['index']
del train_9['index']
del train_10['index']
del train_11['index']
del train_12['index']
del train_13['index']
train_normal.reset_index(inplace=True)
train_dos.reset_index(inplace=True)
train_r2l.reset_index(inplace=True)
train_u2r.reset_index(inplace=True)
train_probing.reset_index(inplace=True)
test_normal.reset_index(inplace=True)
```

```
test_dos.reset_index(inplace=True)
test_r2l.reset_index(inplace=True)
test_u2r.reset_index(inplace=True)
test_probing.reset_index(inplace=True)
del train_normal['index']
del train_dos['index']
del train_r2l['index']
del train_u2r['index']
del train_probing['index']
del test_normal['index']
del test_dos['index']
del test_r2l['index']
del test_u2r['index']
del test_probing['index']
```

Додаток Б
(Обов’язковий)
Копії публікацій

д.т.н., проф. Савенко О.С. (ХмНУ)

к.т.н., доц. Муляр І.В.

(ХмНУ)Любінецький Д.

В. (ХмНУ)

СИСТЕМА ЗАХИСТУ КОРПОРАТИВНОЇ МЕРЕЖІ ОРГАНІЗАЦІЇ ВІДІНСАЙДЕРСЬКИХ АТАК

Розглянемо поняття інсайдерської загрози. Даний тип загроз вважаються внутрішніми тому, що виходить від людей, які з допомогою свого привілейованого доступу до мережі наносять для системи певну шкоду. Така шкода може бути як і умисною, так і ненавмисною . Для того, щоб уберегтися від цих загроз повинна бути визначена комплексна система, яка зможе визначити та нейтралізувати загрозу без шкоди для мережі.

Багато сучасних підприємств при побудові своєї мережі використовують системи виявлення вторгнень або IDS. Це програмний або апаратний засіб, який найчастіше використовують підприємства і фірми для визначення загроз по відношенню до комп'ютерної системи. Алгоритми визначення загроз в таких засобах будуються на основі наперед заданих правил (сигнатур) або евристичного аналізу трафіку. В останні роки, для евристичного аналізу використовують нейронні мережі.

Більшість таких систем вважаються найбільш ефективними проти зовнішніх загроз, але не зовсім так. Мною було запропоновано використовувати IDS для запобігання або принаймні зменшення внутрішніх атак і загроз. Однак налаштування IDS для виявлення внутрішніх атак може бути складним. Частина проблеми полягає в створенні якісного набору

правил для внутрішньої IDS. Причина по якій набір правил має відрізнятися, полягає в тому, що різні користувачі мережі потребують різного обсягу доступу до різних служб, серверів і систем для своєї роботи. Системи, які можна розгорнути для допомоги в боротьбі з інсайдерськими атаками, включають системи виявлення вторгнень у мережу NIDS, NNIDS, HIDS, системи виявлення вторгнень на основі аномалій та аналітично повноважена розподілена система виявлення вторгнень (dIDS). Кожна з цих систем має право на своє використання в мережі, а також свої переваги та недоліки.

УДК 004.056:004.852

АНТОНІНА
КАШТАЛЬЯН
Хмельницький національний університет
e-mail: antonina.kashtalian@gmail.com
ДЕНИС ЛЮБІНЕЦЬКИЙ
Хмельницький національний університет
e-mail:kiberplayer@gmail.com

РОЗПОДІЛЕНА САМООРГАНІЗОВАНА СИСТЕМА ПРОГНОЗУВАННЯ ЗЛОВМИСНОЇ АКТИВНОСТІ В КОМП'ЮТЕРНИХ МЕРЕЖАХ

У роботі розроблено самоорганізовану систему прогнозування зловмисної активності в комп'ютерній мережі згідно алгоритмів роботи глибокого навчання. Крім того, було представлено нову самоорганізовану інкрементну нейронну мережу під назвою FG-SOINN написану мовою програмування Python. У SOINN видалення вузлів і ребер визначається двома параметрами, які потрібно оптимізувати для кожної наявної програми за допомогою перехресної перевірки або подібних підходів повторної вибірки. FG-SOINN усуває цей недолік, розглядаючи видалення вузлів і ребер як невід'ємну частину процесу навчання. Було сформульовано три концепції для формування «сміттєвого забуття»: час простою, надійність і корисність завдяки чому мережа видаляє вузли та ребра. Така мережа базується на концепті «навчання без вчителя» і може працювати із штучними та реальними даними і, навіть, за раптових або повторюваних відхилень.

Keywords: система виявлення вторгнень, аналіз трафіку, глибоке навчання, нейронні мережі, зловмисна активність, самоорганізована система.

ANTONINA KASHTALIAN,
DENYS LIUBINETSKYI
Khmelnytskyi National University

DISTRIBUTED SELF-ORGANIZED SYSTEM FOR PREDICTING MALICIOUS ACTIVITY IN COMPUTER NETWORKS

In this article, a self-organized computer network protection system based on deep learning algorithms was considered In addition, a new self-organizing incremental neural network called FG-SOINN written in the Python programming language was presented. In the SOINN, node and edge removal is defined by two parameters that need to be optimized for each existing program using cross-validation or similar resampling approaches. FG-SOINN overcomes this drawback by treating node and edge removal as an integral part of the learning process. Three concepts were formulated to form "garbage oblivion": idle time, reliability, and utility by which the network removes nodes and edges. Such a network is based on the concept of "learning without a teacher" and will work both with artificial and real data and even with sudden or repeated deviationst.

Keywords: intrusion detection system, traffic analysis, deep learning, neural networks, malicious activity, self-organized system.

Вступ

З безперервним зростанням загроз і атак, є досить складним завданням точно і своєчасно виявити зловмисну активність у комп'ютерних мережах. На сьогодні запропоновано багато принципів, методів, систем до виявлення мережевих вторгнень. Однак вони стикаються з критичними проблемами через постійне збільшення нових загроз, які поточні системи не розуміють.

Мережева активність означає взаємодію різних комп'ютерів для досягнення певних цілей. Зловмисна діяльність стає все більшою проблемою в Інтернеті. Спамери використовують скомпрометовані комп'ютери для надсилання шкідливих даних[1]. Знання комп'ютерні атаки може допомогти передбачити таку активність [2].

Незловмисна діяльність, також, викликає занепокоєння: засоби відстеження реклами та мережі доставки вмісту можуть взаємодіяти з багатьма комп'ютерами. Дослідження доброякісної активності допомагають встановити базову лінію [3] або охарактеризувати його зростання. На жаль, важко зрозуміти масштаби цих потенційних загроз через децентралізацію Інтернету, тому виявлення зловмисної активності в комп'ютерних мережах стає досить складним завданням. Крім того, через часті кібератаки, можна спостерігати тенденцію до того, що вони стають дедалі якіснішими та кваліфікованими. Нездатність запобігти або виявити такі вторгнення може мати серйозні наслідки для користувачів такої мережі. Для запобігання впливу зловмисної активності потрібна система, яка буде розпізнавати шаблони мережевого підключення, щоб класифікувати відомі та невідомі вторгнення, але також вимагає періодичного перенавчання, щоб підтримувати продуктивність на високому рівні.

Постановка проблеми вирішення задач прогнозування зловмисної активності

Зловмисні кібератаки створюють серйозні проблеми безпеки, які потребують нової, гнучкої та більш надійної системи виявлення вторгнень (IDS). IDS — це інструмент проактивного виявлення вторгнень, який використовується для автоматичного виявлення та класифікації вторгнень, атак або порушень політик безпеки. Більшість методів виявлення зловмисних дій, запропонованих у літературі, є методами, заснованими на правилах (збіг сигнатур) і методами прогнозованого моделювання (виявлення аномалій) [4], [5]. Методи, засновані на правилах, зазвичай використовують відомі зловмисні дії як базову лінію для порівняння з новими поведінками, які, як відомо, вказують на порушення безпеки [4]. Зазвичай це досягається шляхом вбудовування евристик для пошуку відомих шаблонів (сигнатур) у мережі та/або даних аудиту. Однак розробити сценарії зловмисної діяльності, які охоплюють усі шаблони та/або невидимі шаблони (тобто атаки нульового дня), досить складно. Крім того, зловмисники можуть знати про евристики виявлення, які вже використовуються механізмом, і намагатися їх уникнути. Тому, потрібні більш надійні методи, які можуть адаптуватися протягом роботи задля попередження зловмисної активності.

Для забезпечення захисту мережі системи потрібно вирішити три важливі проблеми, що пов'язані з безпекою мережі.

1. Перша проблема пов'язана зі швидким збільшення обсягу мережевих даних. Це зростання пов'язано з використанням Інтернету речей (IoT) [6], хмарних сервісів та стрімкого зростання мережі пристроїв. Удосконалення методів аналізу даних включає підвищення швидкості та надійності процесу аналізу.
2. Друга проблема полягає в тому, що більш точне відстеження та інтерпретація значно підвищує якість висновків. Аналіз NIDS (Network Instruction Detection System) вимагає більш контекстно-специфічних спостережень, які підкреслюють більш абстрактні спостереження та спостереження вищого рівня. Усі зміни поведінки мають бути відстежуваними до окремого користувача, версії операційної системи або протоколів певних програм.
3. Третя загроза сучасних мереж — це різноманітність протоколів і масивна передача даних у сучасних мережах. В цьому випадку, існує надзвичайно високий рівень складності, коли намагаються відрізнити ненормальну поведінку від нормальної. Це збільшує ймовірність ненадійних і суперечливих даних та збільшує потенціал впливу вразливостей нульового дня.

Алгоритми інкрементного навчання дозволяють класифікатору з часом уточнювати та вдосконалювати свої можливості (оскільки він обробляє все більшу кількість вхідних даних) на відміну від автономного або пакетного алгоритму навчання, де передбачається, що класифікатор піддається впливу вхідних даних у пакеті. Динаміка мережевих даних значно змінюється з часом, і застосування статичних навчених моделей поступово погіршує продуктивність виявлення, роблячи алгоритми навчання з вчителем непридатними для IDS.

Системи виявлення вторгнень можна розділити на категорії залежно від використовуваного методу виявлення. Відповідно до досліджень в [6], методології виявлення класифікуються на дві категорії: на основі сигнатур і на основі аномалій. Методологія виявлення згідно сигнатур використовує шаблони, попередньо визначені відомими атаками, і поширюється як набір сигнатур. Потім сигнатури порівнюються з шаблонами, виявленими в мережевому трафіку, щоб виявити можливі атаки. Незважаючи на ефективність щодо відомих загроз, такий метод не може виявити або запобігти невідомим атакам і не може підтримувати й оновлювати сигнатури для відомих або нещодавно виявлених атак. Навпаки, виявлення на основі аномалій зазвичай встановлює базовий/нормальний рівень, використовуючи статистично значущий трафік.

Залежно від використовуваної техніки аналізу даних процес навчання/тестування може використовувати або класифікацію, або кластеризацію. Дослідження щодо вдосконалення методів класифікації систем виявлення вторгнень, здебільшого зосереджені на оцінці альтернативних рішень для основного аналізу, включаючи нейронні мережі [7] [8], нечітку логіку [8] [9], генетичні алгоритми [10] та опорні векторні машини [11]. У той же час методи кластеризації в основному використовують k-середні та алгоритми виявлення викидів [12], [13]. Що стосується впровадження, [14] продемонстрував ефективність використання TensorFlow для аналізу набору даних NSL-KDD і виявлення шкідливого трафіку з точністю 96%. Як було підкреслено попередніми дослідженнями [6],

SOINN і поступове навчання дійсно є дуже ефективними методами для вирішення проблем виявлення зловмисної активності.

Під час еволюції IDS стало очевидним, що гібридний підхід дає кращі результати завдяки його здатності розрізняти типи та види мережевих атак, використовуючи спочатку статистичні підходи, щоб покращити формулювання та форматування вхідних даних, а потім застосувати їх до механізму на основі штучного інтелекту. Коли механізм виявлення визначає, чи є трафік частиною атаки, пакети можуть бути зареєстровані або скинуті та частково передані на пристрої-одержувачі.

Постановка задачі

Із збільшенням кількості та розмірів даних потрібні алгоритми навчання, щоб ефективно працювати з великою кількістю сигналів. Крім того, набагато складнішим завданням для навчання без вчителя є ефективне та стійке вивчення даних із розподілів, у яких існують дані з шумами. Основна складність полягає в тому, що алгоритми навчання не мають попередніх знань про розподіл даних у цілому. Таким чином, при надходженні перших кількох даних певного розподілу, кількість даних недостатня для представлення всього розподілу. Наразі алгоритми навчання не можуть визначити, чи є ці дані шумними чи нормальними. Отже, для кожної ітерації існуючі методи (такі як самоорганізуюча карта (SOM) [15], нейронний газ (NG) [16] тощо) мають реагувати на нові дані та оновлювати вагові вектори відповідних нейронів, що зазвичай викликає критичне відхилення відображення топології в результаті. Інша проблема полягає в тому, що в більшості самоорганізованих нейронних мереж «зростаючого типу», таких як зростаючий нейронний газ (GNG) [17], число нейронів буде постійно збільшуватися внаслідок зростання стратегії їх визначення. Велика кількість нейронів збільшує обчислювальні витрати на пошук нейронів-переможців на кожній ітерації, що робить процедуру навчання неефективною. Тому, для вирішення таких задач потрібен алгоритм який буде уникати цих проблем, що в значній мірі покращить ефективність самоорганізованої системи.

Задача дослідження полягає у створенні розподіленої самоорганізованої системи прогнозування зловмисної активності, а саме систему виявлення вторгнень на основі штучного інтелекту та модифікованого модуля самоорганізованої інкрементної нейронної мережі. Такий модуль повинен використовувати структуру техніки «сміттєвого забуття» на основі концепцій часу простою, надійності та корисності.

Реалізація поставленої задачі дозволить визначити, спроектувати та реалізувати систему прогнозуання зловмисної активності шляхом використання нейронних мереж та методів SOIIN.

Архітектура самоорганізованої системи прогнозування

Для правильного функціонування системи роботи, спрямованого на визначення зловмисної активності, потрібно визначити її архітектуру.

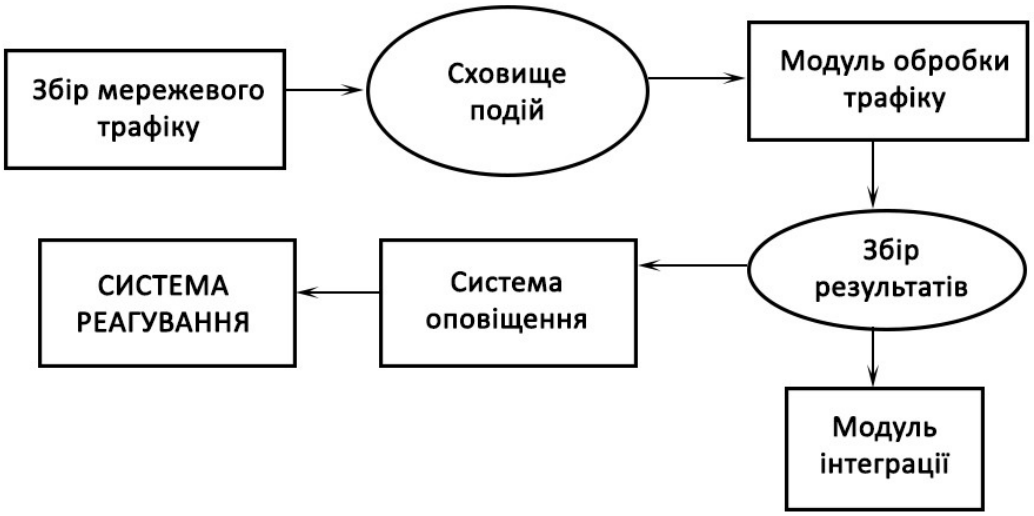


Рисунок 1 –Архітектура системи IDS на основі FG-SOINN

Така система буде складатися з наступних компонент:

Збір мережевого трафіку необхідний для збору всієї наявної інформації з пристроїв мережі. Крім того виконує функцію перетворення вихідного мережевого трафіку в необхідний вигляд (обрахування і визначенння

необхідних параметрів) і записування даних в сховище подій. Сховище вихідний подій визначене системою як місце для зберігання інформації, що надалі аналізується системою на наявність шкідливого трафіку і наявність мережевої атаки. Кожен запис у сховищі буде зберігати інформацію про потоки даних за параметрами, які необхідні для модуля обробки трафіку. Модуль обробки трафіку являє собою систему яка працює на алгоритмі самоорганізованих інкрементних нейронних мереж, розроблених у даній роботі. Він виконує аналіз трафіку кожного запису про потік, що зберігається в базі даних вихідних подій за допомогою алгоритмів, що використовують методи машинного навчання. В кінцевому результаті для кожного конкретного запису визначається тип з'єднання: нормальне або атака, а також вид самої атаки у випадку виявлення такої. Результати аналізу даних будуть занотовані в базу даних результатів. Місце збору результатів являє собою відокремлену базу даних, де зберігатимуться виявлені аномалії. Дана має використовуватися системою сповіщень, а також модулем сумісності для вилучення результатів аналізу. Система реагування буде виконувати функції «сміттяра». Кожен визначений системою шкідливий запис буде видалятися, не визначені записи, тобто ті записи, що не є і не нормальними і не шкідливими, буде повертатися у модуль обробки трафіку для подальшої повторної обробки. Надалі знову система звертається до модуля збору мережевого трафіку для подальшого циклу роботи. Модуль інтеграції являє собою API для можливості інтегрування з системами реагування, інтерфейс системної взаємодії за допомогою http-запитів.

Основна частина

Самоорганізована інкрементна нейронна мережа (SOINN) — це механізм неконтрольованого навчання (або навчання без вчителя) для немаркованих даних. Самоорганізовані нейронні мережі відіграли важливу роль у сфері неконтрольованого навчання протягом останніх кількох десятиліть. Неконтрольоване навчання(навчання без вчителя) має дві основні цілі: кластеризацію та вивчення топології даних. Метою кластеризації є розділення даного набору даних на кілька кластерів, де кожна пара даних в одному кластері має більшу схожість, ніж у двох різних кластерах [18]. З іншого боку, навчання топології даних можна описати наступним чином: враховуючи розподіл даних високої розмірності, потрібно спроектувати вхідні дані в топологічну структуру, в якій дані у вхідному просторі проектуються топологічні суміжні одиниці [19]. Останнім часом ця техніка широко застосовується для інтелектуального аналізу даних, векторного квантування, розпізнавання образів, комп'ютерного зору та багатьох інших суміжних галузей [20].

Як неконтрольований інкрементний метод кластеризації, SOINN пропонує відносно високу швидкість обчислення з низькою обчислювальною вартістю. Крім того, складність мережі та розмір SOINN контролюються та стабілізуються за допомогою техніки «збирача сміття». Техніка визначає параметр виснаження, який називається віком, який представляє часові рамки, після яких вузли періоду буде видалено, якщо вони не оновлюються протягом зазначеного часу, і таким чином динамічно усуватиме шум у даних. Ця властивість робить його привабливим для динамічних середовищ, де потрібне тривале навчання. Щоб забезпечити її масштабованість під час розширення, розмір і зростання мережі контролюються параметром n, де кілька пар SOINN використовуватимуться як контрольований метод кластеризації. SOINN, як підсумовано на рисунку 1, ініціалізує мережу порожнім набором вузлів, потім додає перші два вузли до списку, використовуючи вагові вектори як два вхідних вектора [21]. Після ініціалізації нейронна мережа знаходить для кожного вхідного вектора найближчий вузол (переможець) і другий найближчий вузол (другий переможець) вхідного вектора, вимірюючи відстань S1 і S2 від кожного входу до кожного вузла за допомогою рівнянь (1) і (2). Дана формула є загальною формулою для вимірювання відстані між шарами.

$$s_1 = argmin_{C \in A} dist(x, w_c) \tag{1}$$

$$s_2 = argmin_{C \in A - \{s_1\}} dist(x, w_c) \tag{2}$$

Якщо вхідний вектор належить до того ж кластеру, що й переможець або другий переможець на основі відстаней, обчислених за пороговим критерієм подібності, SOINN оновлює ваговий вектор вузла та його сусідів із ваговим вектором вхідного значення та з'єднує його з вузол ребром. Якщо вхідний вектор не належить до того самого кластера переможця або другого переможця, механізм додає новий вузол до мережі.

Самоорганізуючі інкрементні нейронні мережі (SOINN) охоплюють сімейство нейронних мереж, спільним для яких є те, що вони знаходять топологічне відображення вхідних даних у мережеву структуру шляхом конкурентного навчання (21).

Можна сказати, що SOINN відображає p -вимірний вхід $x = x_1 x_2 \dots x_p$ де $x_i \in i^{-M}$ значенням ознаки для окремого вузла в неорієнтованому графі. Відображення відповідає точці в p - вимірному просторі ознак. Навчання в SOINN означає адаптацію топологічної карти: вузли можуть переміщатися, об'єднуватися з іншими вузлами, залишатися одиночними або видалятися, а межі між вузлами можна створювати або видаляти. Вузол треба

розглядати як мікрокластер вхідних випадків, які знаходяться близько один до одного. Ребра можна розглядати як консолідовані зв'язки між пов'язаними вузлами, наприклад, вузлами, що належать одному (макро)кластеру.

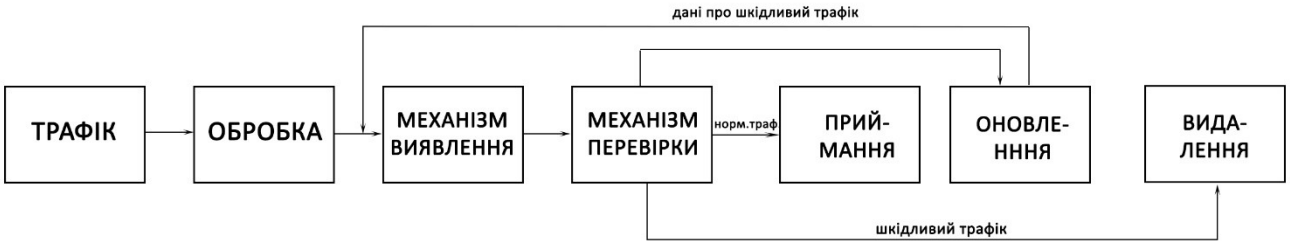


Рисунок 2 – Інформаційні потоки системи на основі FG-SOINN

На рисунку 2 зображено інформаційні потоки тієї частини системи, в якій досліджується трафік на предмет його класифікації. Спершу вхідний трафік зібраний системою подається на модуль попередньої обробки. Цей модуль фіксує та обробляє вхідний трафік у реальному часі. Перехоплені TCP-з'єднання обробляються для вилучення відповідних функцій, які стають доступними як вхідний вектор для механізму виявлення – FG-SOINN. Для дослідження перевірка була виконана з використанням назви 41-го атрибуту набору даних NSL-KDD, і вся інформація про атрибути доступна в [22]. Структура базується на передумові, що характеристики TCP з'єднання вказують на те, чи з'єднання визначається як атака чи ні, і навіть якщо воно визначається як атака, то яким типом є ця атака. Далі інформація подається на механізм виявлення зловмисної активності, який після обробки передає інформацію про трафік на модуль механізму перевірки даних. Далі модуль перевірки визначає покращення та підвищення точності системи шляхом підтвердження передбаченої мітки. Саме на цьому етапі відбувається розподілення трафіку на шкідливий та нормальний. Після підтвердження того, що трафік є нормальним його передають користувачу для подальшої роботи. Але якщо система підтвердила виявлення шкідливого трафіку, модуль перевірки даних пересилає невдалі прогнози до модуля оновлення і видаляє з набору даних трафіку.

Модуль оновлення системи працює у дві фази: активна фаза та фаза оновлення. На фазі реального часу він приймає рішення на основі того, які його можливості були в той час, і на фазі оновлення, коли модуль оновлення оновлює систему за допомогою невдалих прогнозів, щоб покращити її можливості. Фази можуть виконуватися паралельно, якщо це необхідно у виробництві, або чергуватися відповідно до мережевого трафіку.

Основна концепція запропонованого алгоритму полягає в поступовому створенні механізму захисту мережі. На початковому етапі механізм виявлення навчається, використовуючи відносно невелику вибірку мережевих даних, достатню для базового захисту мережі. Згодом, коли стає доступним більше мережевих даних, механізм поступово оновлюється вхідними даними класів, які йому не вдалося виявити, щоб удосконалити та розширити свої захисні можливості. Щоб підтримати процес навчання, механізм перевірки вирішує, чи не вдалося прийняти рішення. Щоб розширити свої можливості, основний механізм виявлення повинен бути в змозі класифікувати мережеві дані за багатьма класами, не тільки про те, чи є з'єднання атакою чи ні, але й про тип атаки, а отже, пропонувати рішення для багатокласової проблеми поступового навчання.

Структура системи, використовує концепцію, що лежить в основі n-SOINN [23], яка модифікувала оригінальний SOINN для використання кількох пар SOINN з використанням підходу контрольованої кластеризації. n-SOINN використовує дві важливі модифікації: глобальний параметр, який контролює топологію мережі та використовує квадрат (а не базову) евклідову відстань для обчислення відстані між входом і вузлами [24]. Щоб контролювати кількість вихідних вузлів мережі, де різниця в тому, наскільки точно буде створена стиснена інформація, вводиться параметр з іменем n. Цей параметр диктує, що будь-який перший вузол-переможець, який виграє більше n разів, призначає перемогу другому вузлу-переможцю. Якщо другий вузол-переможець також має більше ніж n разів виграшу, генерується новий вузол. Для n=0 мережа поводиться точно так само, як вихідний SOINN. Встановлення дуже високих значень n зменшує кількість створених стабільних вузлів і надає перевагу лише популярним, що швидше за все, призведе до невеликої точності. Евклідова відстань, використана в оригінальному SOINN, була призначена для цілей єдиного SOINN для реалізації завдання інкрементного навчання без контролю (без вчителя). Використання квадрата евклідової відстані в n-SOINN дозволяє вимірювати відстань між вузлами для різних SOINN; старіння, вбудоване в забувач сміття (forgetting garbage або FG) SOINN дозволяє усувати вузли вхідних даних, коли вони стають непотрібними або не популярнимим. Основний модуль системи складається з двох основних частин: кластеризації та класифікатора. Блок кластеризації містить пару n-SOINN, які використовуються для кожного класу для стиснення інформації,

отриманої від TCP-з'єднань модулем попередньої обробки, і досягнення інкрементного навчання. Частина класифікації приймає вихідні дані вузлів n-SOINNs, створює вхідні дані для класифікатора SVM для кожного класу виконання попередньої класифікації. Згодом m найвищих класів пар класифікації кожного класифікатора, відсортованих за оцінкою, потім класифікуються багатокласовою SVM для кінцевого рішення (m визначається користувачем відповідно до доступних класів). Оцінка базується на відстані зразків до розділової гіперплощини.

Експериментальне дослідження

Оцінка отриманої структури була виконана на наборі даних NSL-KDD [25], який є вдосконаленою версією відомого набору даних KDD'99. Незважаючи на свій вік, набір даних все ще є де-факто альтернативою для методів і інструментів порівняльного аналізу, які спрямовані на забезпечення ефективних систем виявлення вторгнень [26]. З огляду на його широке використання, що полегшує надання довідкового аналізу, було прийнято цей набір даних для початкового тестування впровадженого методу. Для негативного SOINN було використано n=2, а для позитивного SOINN n=100. Оскільки набір даних включав суміш звичайного трафіку та чотирьох атак, для цього експерименту було створено п'ять бінарних класів – звичайний клас і чотири класи атак (відмова в обслуговуванні, зондування, R2L і U2R).

Використаний набір даних включав дві різні підмножини, одну з 125973 записами, а іншу з 22544 записами (рис.4).

```
rand_int = random.randint(1, len(train) - 1)
x1 = train.iloc[rand_int].values
rand_int = random.randint(1, len(train) - 1)
x2 = train.iloc[rand_int].values
rand_int = random.randint(1, len(train) - 1)
x3 = train.iloc[rand_int].values

s = SF_SOINN(x1, x2, x3, max_edge_age=100, iter_lambda=100)

xs, n_nodes, n_edges, n_del_nodes, n_del_edges = train_phase(mode

Час навчання: 21 хв 29 сек
Вхідні дані оброблено: 125972
Кількість вузлів: 1076
Кількість ребер: 4712
```

Рисунок 4 – Обробка даних NSL-KDD

Друга підмножина не з того ж розподілу ймовірностей, що й перша, і також містить певні типи атак, яких немає в першій. Для базової оцінки та для того, щоб показати, що досягається поступове навчання, ми розділили першу підмножину на п'ять менших підмножин, які використовуються для раундів тестування/оновлення, і використали другу підмножину з 22543 записами для початкового навчання. Слід зазначити, що після початкового навчання для кожного раунду оновлення підмножина перевіряється на навченій FG-SOINN, і лише невдалі прогнози повертаються в систему.

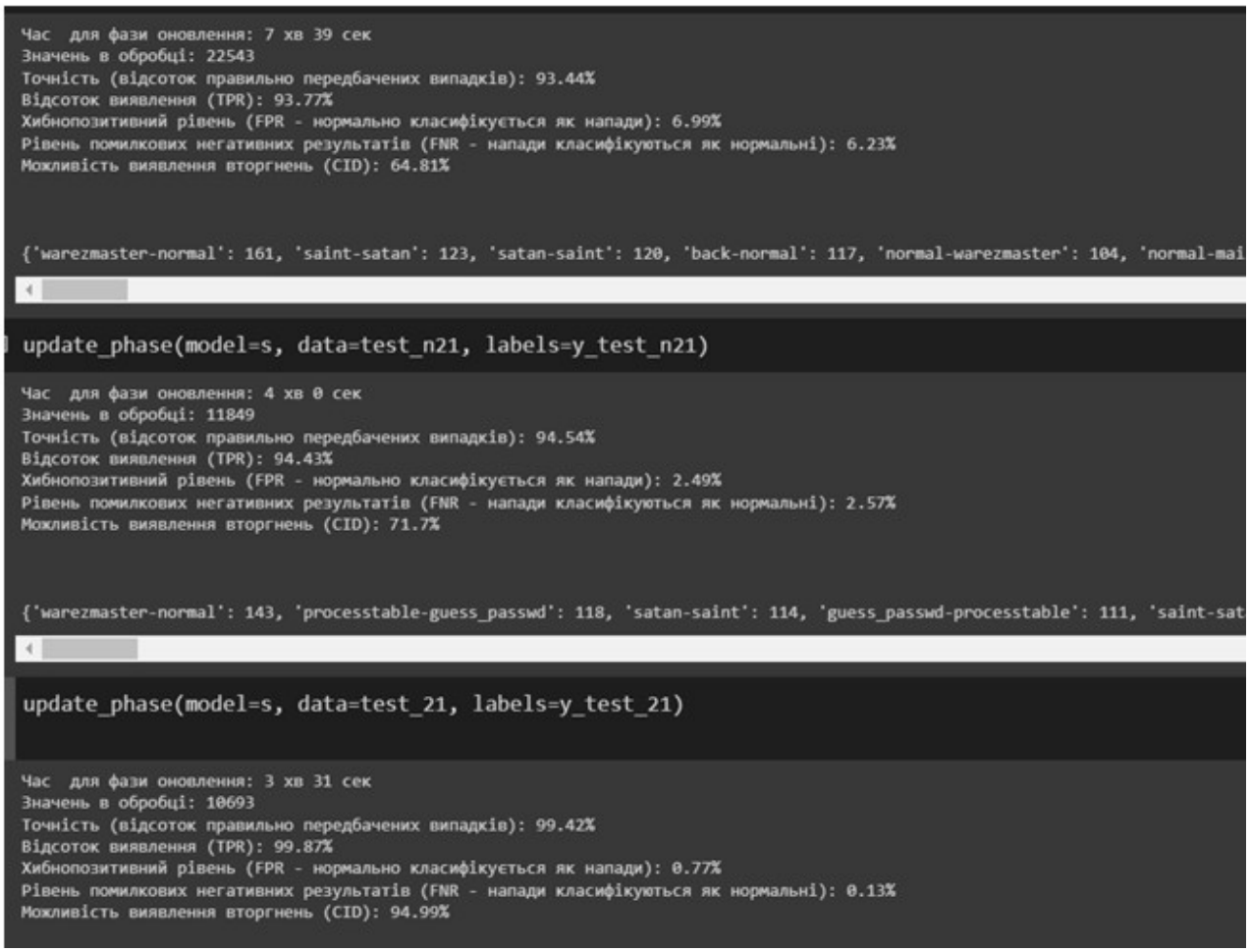


Рисунок 5 – Результати тренування системи після трьох раундів

Таблиця 1

Результати тренування FG-SOINN

Фаза оновлення	Точність [%]	Час [s]	Зразки
1	93.44	459	22543
2	94.54	699	37392
3	99.42	910	48085

Результати показані на рисунку 5, вказують на те, що структура може використовувати поступове навчання для досягнення суттєвих покращень. Така система визначає 5 основних показників:

- точність або відсоток виявлення випадків;
- справжній позитивний коефіцієнт (TPR) або коефіцієнт виявлення (DR): співвідношення між кількістю правильно передбачених атак і загальною кількістю атак, також називається коефіцієнтом виявлення (DR);
- рівень хибнопозитивних результатів (FPR): співвідношення між кількістю звичайних випадків, неправильно класифікованих як атаки, та загальною кількістю звичайних випадків;
- хибнонегативний показник (FNR): не вдалося визначити аномалію та класифікувати як нормальний;
- можливість виявлення вторгнень (CID): співвідношення спільної інформації між входом і виходом та ентропією входу.

Дані показники визначаються як метрики оцінки алгоритму роботи. Зразки, наведені в таблиці 1 є накопиченими зразками з попереднього раунду. Наприклад, для початкового навчання використовувався набір даних, що містив 22543 записи, а для першого раунду оновлень було використано 11849 записи, тобто кількість невдалих прогнозів для першої підмножини з п’яти накопичується до 37392. Точність навчання покращується з кожною новою фазою, що підтверджує твердження про ефективність запропонованої структури.

Висновки

В роботі запропонована самоорганізована система прогнозування зловмисної активності на основі інкрементного навчання. Виконується вона з допомогою системи виявлення вторгнень яка навчається з допомогою нейронних мереж. Запропонована структура та результати вказують на те, що така пропозиція може адаптуватися до природи динамічного профілю мережевих даних як для звичайних категорій, так і для категорій атак. Система використовує менше ресурсів, є швидшою і має вищу швидкість виявлення, ніж методи навчання з вчителем. Забезпечуючи систему невдалими прогнозами, ми не лише досягаємо поступового навчання з багатообіцяючою точністю, але й ефективну структуру, тобто замість того, щоб подавати їй повний набір даних, FG-SOINN зберігає лише частину набору даних, роблячи дану структуру дуже хорошим кандидатом для систем масштабування. Незважаючи на те, що час навчання системи збільшується зі збільшенням вхідних даних оновлення, режими оновлення системи та робочий режим можуть або працювати паралельно (одночасно), або режим оновлення може перемикатися, коли робоча фаза неактивна (тобто немає вхідних даних для виявлення). Таким чином вона набуває здібностей шляхом вивчення нових вхідних даних або невдалих класифікацій. Саме через ці здібності система буде вважатися самоорганізованою.

Програма була реалізована з допомогою середовища Jupiter Notebook та мови програмування Python. Така система здатна прогнозувати зловмисну активність та одночасно навчатися на невдалих попередніх спробах, що робить її більш ефективною перед штучними та реальними даними.

Література

1. Huang D. Y. et al. Tracking ransomware end-to-end //2018 IEEE Symposium on Security and Privacy (SP). – IEEE, 2018. – С. 618-631.

2. J. Czyz et al., “Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks,” in Proc. ACM Internet Meas. Conf., Vancouver,BC, Canada, Nov. 2014, pp. 435–448.

3. E. Wustrow, M. Karir, M. Bailey, F. Jahanian, and G. Houston, “Internetbackground radiation revisited,” in Proc. 10th ACM Internet Meas. Conf. ,Melbourne, VIC, Australia, Nov. 2014, pp. 62–74

4. Chattopadhyay M. Modelling of intrusion detection system using artificial intelligence—evaluation of performance measures //Complex System Modelling and Control Through Intelligent Soft Computations. – Springer, Cham, 2015. – С. 311-336.

5. Constantinides C. et al. A novel online incremental learning intrusion prevention system //2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS). – IEEE, 2019. – С. 1-6.

6. Gokhale P., Bhat O., Bhat S. Introduction to IOT //International Advanced Research Journal in Science, Engineering and Technology. – 2018. – Т. 5. – №. 1. – С. 41-44.

7. Santikellur P. et al. Optimized multi-layer hierarchical network intrusion detection system with genetic algorithms //2019 2nd International Conference on new Trends in Computing Sciences (ICTCS). – IEEE, 2019. – С. 1-7.

8. J. J. Stephan, S. Mohammed, and M. K. Abbas, “Neural Network Approach to Web Application Protection,” Int. J. Inf. Educ. Technol., vol. 5, no. 4, 2015.

9. S. N. Shiaeles, V. Katos, A. S. Karakos, and B. K. Papadopoulos, “Real time DDoS detection using fuzzy estimators,” Comput. Secur., vol. 31, no. 6, pp. 782–790, 2012

10. Varzaneh Z. A., Kuchaki Rafsanjani M. Intrusion detection system using a new fuzzy rule-based classification system based on genetic algorithm //Intelligent Decision Technologies. – 2021. – Т. 15. – №. 2. – С. 231-237.

11. Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, “An efficient intrusion detection system based on support vector machines and gradually feature removal method,” Expert Syst. Appl., vol. 39, no. 1, pp. 424–430, 2012.

12. M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “A multi-step outlier-based anomaly detection approach to network-wide traffic,” Inf. Sci. (Ny)., vol. 348, pp. 243–271, 2016

13. T. Bakhshi and B. Ghita, “User traffic profiling,” in 2015 Internet Technologies and Applications (ITA), 2015, pp. 91–97.

14. L.-D. Chou et al., “Classification of Malicious Traffic Using TensorFlow Machine Learning,” in International Conference on Information and Communication Technology Convergence, 2018, pp. 186–190.

15. Zhang J. et al. Intelligent fault diagnosis of rolling bearings using variational mode decomposition and self-organizing feature map //Journal of Vibration and Control. – 2020. – Т. 26. – №. 21-22. – С. 1886-1897.

16. E. J. Palomo and E. López-Rubio, "The Growing Hierarchical Neural Gas Self-Organizing Neural Network," in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 9, pp. 2000-2009, Sept. 2017, doi: 10.1109/TNNLS.2016.2570124.

17. H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y.Tung, “Intrusion detection system: A comprehensive review,” J. Netw. Comput. Appl., vol. 36, no. 1, pp. 16–24, 2013.

18. R. Xu and D. C. W. II, “Survey of clustering algorithms.” IEEE Trans. Neural Netw., vol. 16, № 3, pp. 645–678, 2015

19. T. Martinetz and K. Schulten, “Topology representing networks,” Neural Networks, vol. 7, no. 3, pp. 507–522, 2014

20. K.-L. Du, “Clustering: A neural network approach,” Neural Networks, vol. 23, no. 1, pp. 89–107, 2010.

21. Furao S., Ogura T., Hasegawa O. An enhanced self-organizing incremental neural network for online unsupervised learning //Neural Networks. – 2007. – T. 20. – №. 8. – C. 893-903.

22. Meena G., Choudhary R. R. A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA //2017 International Conference on Computer, Communications and Electronics (Comptelix). – IEEE, 2017. – C. 553-558.

23. P. Kankuekul, A. Kawewong, S. Tangruamsub, and O. Hasegawa, “Online incremental attribute-based zero-shot learning,” 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2012.

24. S. Furao and O. Hasegawa, “An incremental network for on-line unsupervised classification and topology learning,” Neural Networks, vol. 19, no. 1, pp. 90–106, 20016.

25. Cervantes J. et al. A comprehensive survey on support vector machine classification: Applications, challenges and trends //Neurocomputing. – 2020. – T. 408. – C. 189-215.

26. L. Dhanabal and D. S. P. Shantharajah, “A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms,” 2015.

References

1. Huang D. Y. et al. Tracking ransomware end-to-end //2018 IEEE Symposium on Security and Privacy (SP). – IEEE, 2018. – C. 618-631.

2. J. Czyz et al., “Taming the 800 pound gorilla: The rise and decline of NTP DDoS attacks,” in Proc. ACM Internet Meas. Conf., Vancouver,BC, Canada, Nov. 2014, pp. 435–448.

3. E. Wustrow, M. Karir, M. Bailey, F. Jahanian, and G. Houston, “Internetbackground radiation revisited,” in Proc. 10th ACM Internet Meas. Conf. ,Melbourne, VIC, Australia, Nov. 2014, pp. 62–74

4. Chattopadhyay M. Modelling of intrusion detection system using artificial intelligence—evaluation of performance measures //Complex System Modelling and Control Through Intelligent Soft Computations. – Springer, Cham, 2015. – C. 311-336.

5. Constantinides C. et al. A novel online incremental learning intrusion prevention system //2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS). – IEEE, 2019. – C. 1-6.

6. Gokhale P., Bhat O., Bhat S. Introduction to IOT //International Advanced Research Journal in Science, Engineering and Technology. – 2018. – T. 5. – №. 1. – C. 41-44.

7. Santikellur P. et al. Optimized multi-layer hierarchical network intrusion detection system with genetic algorithms //2019 2nd International Conference on new Trends in Computing Sciences (ICTCS). – IEEE, 2019. – C. 1-7.

8. J. J. Stephan, S. Mohammed, and M. K. Abbas, “Neural Network Approach to Web Application Protection,” Int. J. Inf. Educ. Technol., vol. 5, no. 4, 2015.

9. S. N. Shiaeles, V. Katos, A. S. Karakos, and B. K. Papadopoulos, “Real time DDoS detection using fuzzy estimators,” Comput. Secur., vol. 31, no. 6, pp. 782–790, 2012

10. Varzaneh Z. A., Kuchaki Rafsanjani M. Intrusion detection system using a new fuzzy rule-based classification system based on genetic algorithm //Intelligent Decision Technologies. – 2021. – T. 15. – №. 2. – C. 231-237.

11. Y. Li, J. Xia, S. Zhang, J. Yan, X. Ai, and K. Dai, “An efficient intrusion detection system based on support vector machines and gradually feature removal method,” Expert Syst. Appl., vol. 39, no. 1, pp. 424–430, 2012.

12. M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, “A multi-step outlier-based anomaly detection approach to network-wide traffic,” Inf. Sci. (Ny)., vol. 348, pp. 243–271, 2016

13. T. Bakhshi and B. Ghita, “User traffic profiling,” in 2015 Internet Technologies and Applications (ITA), 2015, pp. 91–97.

14. L.-D. Chou et al., “Classification of Malicious Traffic Using TensorFlow Machine Learning,” in International Conference on Information and Communication Technology Convergence, 2018, pp. 186–190.

15. Zhang J. et al. Intelligent fault diagnosis of rolling bearings using variational mode decomposition and self-organizing feature map //Journal of Vibration and Control. – 2020. – T. 26. – №. 21-22. – C. 1886-1897.

16. E. J. Palomo and E. López-Rubio, "The Growing Hierarchical Neural Gas Self-Organizing Neural Network," in IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 9, pp. 2000-2009, Sept. 2017, doi: 10.1109/TNNLS.2016.2570124.

17. H.-J. Liao, C.-H. Richard Lin, Y.-C. Lin, and K.-Y.Tung, “Intrusion detection system: A comprehensive review,” J. Netw. Comput. Appl., vol. 36, no. 1, pp. 16–24, 2013.

18. R. Xu and D. C. W. II, “Survey of clustering algorithms.” IEEE Trans. Neural Netw., vol. 16, № 3, pp. 645–678, 2015

19. T. Martinetz and K. Schulten, “Topology representing networks,” Neural Networks, vol. 7, no. 3, pp. 507–522, 2014

20. K.-L. Du, “Clustering: A neural network approach,” Neural Networks, vol. 23, no. 1, pp. 89–107, 2010.

21. Furao S., Ogura T., Hasegawa O. An enhanced self-organizing incremental neural network for online unsupervised learning //Neural Networks. – 2007. – T. 20. – №. 8. – C. 893-903.

22. Meena G., Choudhary R. R. A review paper on IDS classification using KDD 99 and NSL KDD dataset in WEKA //2017 International Conference on Computer, Communications and Electronics (Comptelix). – IEEE, 2017. – C. 553-558.

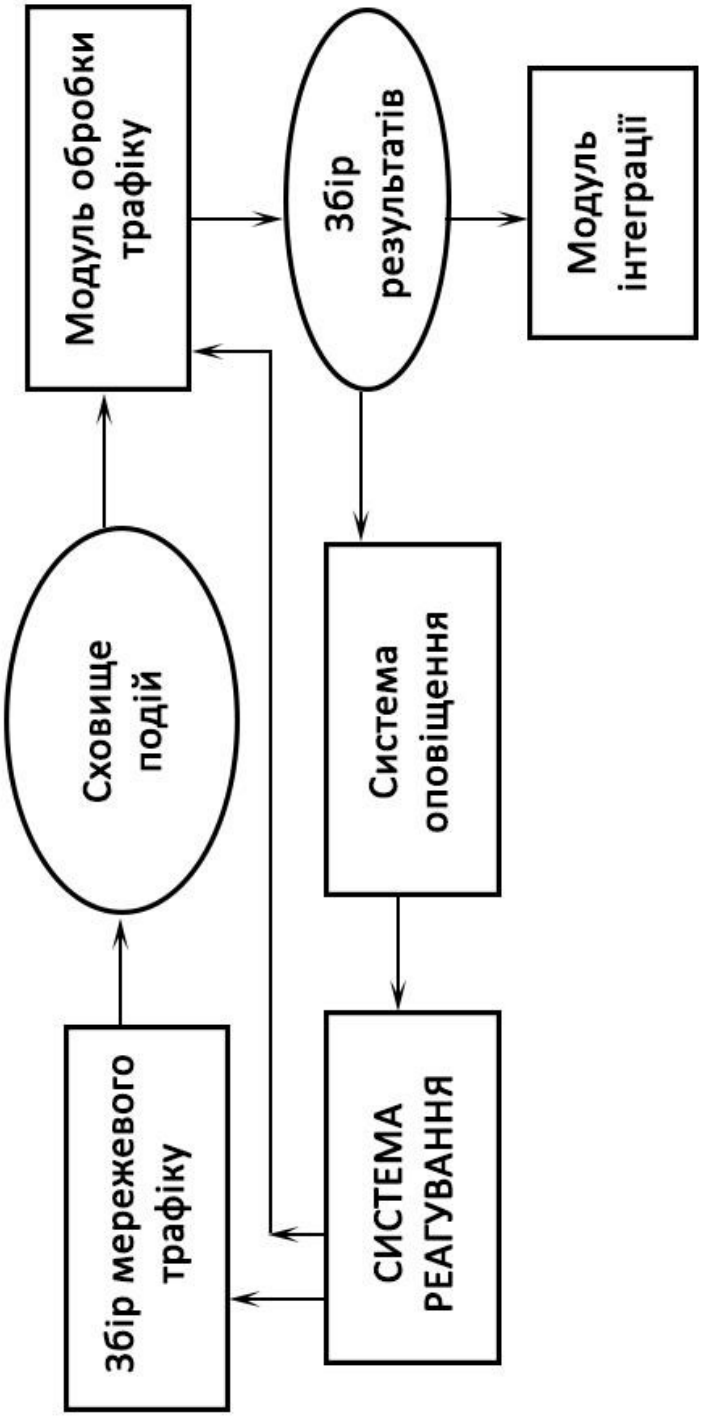
23. P. Kankuekul, A. Kawewong, S. Tangruamsub, and O. Hasegawa, “Online incremental attribute-based zero-shot learning,” 2012 IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2012.

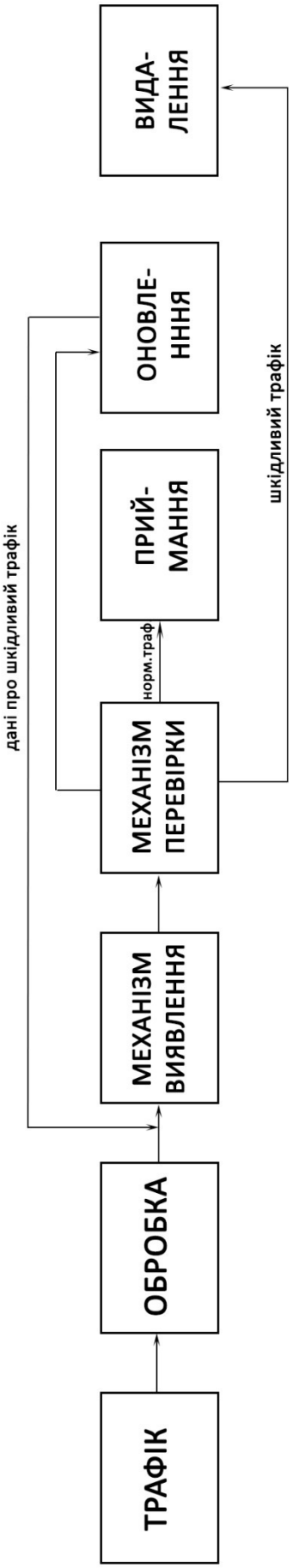
24. S. Furao and O. Hasegawa, “An incremental network for on-line unsupervised classification and topology learning,” Neural Networks, vol. 19, no. 1, pp. 90–106, 20016.

25. Cervantes J. et al. A comprehensive survey on support vector machine classification: Applications, challenges and trends //Neurocomputing. – 2020. – T. 408. – C. 189-215.

26. L. Dhanabal and D. S. P. Shantharajah, “A Study on NSL-KDD Dataset for Intrusion Detection System Based on Classification Algorithms,” 2015.

Додаток В
(Обов’язковий)
Копія графічної частини




ТРАФІК
ОБРОБКА
МЕХАНІЗМ ВИЯВЛЕННЯ
МЕХАНІЗМ ПЕРЕВІРКИ
норм.траф
ПРИЙ-МАННЯ
ОНОВЛЕ-ННЯ
ВИДА-ЛЕННЯ
дані про шкідливий трафік
шкідливий трафік

Fri Dec 09 09:27:10 EET 2022, Мостовий Сергій Володимирович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 0.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 12%**

	Документ		Суммарное совпадение по Базе Данных	
ID: 109163 Название: Розподілена самоорганізована система прогнозування зловмисної активності в комп'ютерних мережах Добавлено в БД: 2022-12-09 Авторы: Любінецький Д.В. Руководители: Савенко О.С. Консультанты: Опоненты:	Символы	Лексемы	Символы	Лексемы
	139918	1063	465 (0%)	11 (1%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы

UNICHECK

Ім'я користувача:
Кафедра кібербезпеки

ID перевірки:
1013253197

Дата перевірки:
09.12.2022 10:23:17 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
09.12.2022 10:24:34 EET

ID користувача:
100008300

Назва документа: **Магістерська_Любінецький**

Кількість сторінок: **114** Кількість слів: **22984** Кількість символів: **170098** Розмір файлу: **1.67 MB** ID файлу: **1013011665**

5.63%
Схожість

Найбільша схожість: **0.39%** з Інтернет-джерелом (http://www.haselab.info/soinn-e.html)

5.17% Джерела з Інтернету	168	Сторінка 116
0.9% Джерела з Бібліотеки	14	Сторінка 119

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	62

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Розподілена самоорганізована система прогнозування зловмисної активності в комп'ютерних мережах

Автор: Любінецький Денис Володимирович

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: Програмування та захист комп'ютерних систем і мереж

Керівник: Савенко Олег Станіславович, д. т. н, проф.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи.	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 5.63% % що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О.С. Савенко

Завідувач кафедри КбКСМ, гарант ОП _____ Ю.П. Кльоц

Дата: 09.12.2022

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Любінецького Дениса Володимировича
ПІБ здобувача вищої освіти

студента ФІТ, 2 курсу, групи КІм-21-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

__5.12.2022__
дата

підпис

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «магістр»

Студент Любінецький Денис Володимирович

Тема – Розподілена самоорганізована система прогнозування зловмисної активності в комп'ютерних мережах

Спеціальність – 123 Комп'ютерна інженерія

Обсяг кваліфікаційної роботи освітньо-професійного рівня «магістр»:

кількість листів креслень 2; кількість сторінок записки 77

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі було досліджено розподілені системи виявлення зловмисного програмного забезпечення, проектовано і реалізовано систему виявлення вторгнень на основі самоорганізованої нейронної мережі

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині роботи

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Студент у вступі визначив основні питання актуальності систем та актуальності роботи. У першому розділі проведено аналіз систем прогнозування зловмисної активності, та розподілених систем. Другий розділ включає дослідження та формалізацію шкідливого програмного забезпечення та його впливи на комп'ютерну систему. У третьому розділі студент побудував архітектуру проектованої в роботі системи та розглянув алгоритми для системи прогнозування. Четвертий розділ включає опис і реалізацію обраного студентом модифікованого алгоритму та опис програмної реалізації впровадженої системи.

4. Позитивні сторони роботи Кваліфікаційна робота має високу практичну та теоретичну цінність, через використання самоорганізованих систем. Практична цінність досягається ефективною роботою. Теоретична цінність роботи досягається системою яка набуває більше можливостей шляхом вивчення невдалих класифікацій або нових вхідних даних. На основі цієї розроби в подальшому може здійснюватися розробка нового програмного забезпечення та нових програм.

5. Негативні сторони роботи У даній роботі застосовується не найефективніший алгоритм впровадження нейронних мереж для впровадження даних в систему виявлення вторгнень

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно, пояснювальна записка відповідає нормам щодо її оформлення.

7. Відгук про роботу в цілому Загальна характеристика кваліфікаційної роботи заслуговує позитивної оцінки. Весь опрацьований матеріал роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Впроваджений метод є досить актуальним на сьогоднішній день

8. Інші зауваження
При перевірці ефективності для порівняння не враховані інші потужні алгоритми

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Мартинюк Валерій Володимирович, завідувач кафедри АКІТ, д.т.н., проф.

« 07. » грудень 2022. (підпис)