Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп`ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеню вищої освіти – магістр
за освітньо-професійною програмою
«Комп`ютерні науки»

зі спеціальності
122 – Комп`ютерні науки

тема роботи:
«Методи та засоби захисту комп`ютерних мереж»

Виконав студент гр. КН-24м. _____ Назаров Н. М.

Керівник _____ Кумченко Ю. О.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп`ютерних наук і технологій*

Ступінь вищої освіти: *Магістр*

Спеціальність: *122 – Комп`ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедри: *к.т.н. Рубан С.А.*

_____

«15» *травня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу магістра

студентові групи КН-24м *Назарову Назару Максимовичу*

1. **Тема кваліфікаційної роботи:** *«Методи та засоби захисту комп`ютерних мереж»*

затверджено наказом по університету № *257с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *04.12.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 88с., додатки, презентація у Microsoft PowerPoint (23 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-3 доц. Кумченко Ю.О.

_____

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*10.02.25*
2	*Розділ 1*	*15.04.25*
3	*Розділ 2*	*18.07.25*
4	*Розділ 3*	*19.11.25*
5	*Висновки*	*20.11.25*
6	*Оформлення кваліфікаційної роботи*	*25.11.25*
7	*Підготовка презентації та графічного матеріалу*	*28.11.25*
8	*Підготовка доповіді до захисту*	*01.12.25*

6. Дата видачі завдання: *24.12.2024р.*

Керівник _____ / Кумченко Ю. О./

7. Запевнення: *Я, Назаров Назар Максимович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Здобувач _____ / Назаров Н. М./

АНОТАЦІЯ

Назаров Н.М. Методи та засоби захисту комп`ютерних мереж: кваліфікаційна робота магістра: 122 - Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 89 с.

Робота складається зі вступу, трьох розділів, висновків та переліку використаної літератури з 36 позицій. Загальний обсяг роботи становить 88 сторінок, з яких основний зміст викладено на 77 сторінках. Включено 1 таблиця, 5 формул та 27 рисунків.

Об’єктом дослідження є процес захисту комп’ютерних мереж від несанкціонованого втручання та аномальної активності. У роботі розглянуто теоретичні та практичні аспекти розробки інтелектуальної системи виявлення мережевих вторгнень, що використовує методи та алгоритми машинного навчання для ідентифікації аномальної активності та невідомих загроз.

Досліджено існуючі підходи до забезпечення кібербезпеки та технологічний стек для створення таких систем, зокрема, використання мови Python, бібліотек Scikit-learn, TensorFlow/Keras, Pandas для реалізації функціоналу аналізу трафіку.

Новизна роботи полягає у застосуванні архітектури глибокого автокодувальника для реалізації стратегії навчання на одному класі на базі датасету CIC-IDS 2017.

Ключові слова:

Кібербезпека, Захист комп’ютерних мереж, Intrusion Detection System (NIDS), Машинне навчання, Штучний інтелект, Глибоке навчання, Autoencoder, Isolation Forest, Навчання без учителя, Виявлення аномалій, Аналіз мережевого трафіку, TensorFlow, Keras, Python.

ANNOTATION

Nazarov N.M. Methods and means of protecting computer networks: Master's Qualification Thesis: 122 – Computer Science. Kryvyi Rih. Kryvyi Rih National University, 2025. 89 p.

The thesis consists of an introduction, three chapters, conclusions, and a list of 36 references. The total volume of the thesis is 88 pages, of which the main content is presented on 77 pages. It includes 1 table, 5 formulas, and 27 figures.

The object of research is the process of protecting computer networks from unauthorized interference and abnormal activity. The paper considers theoretical and practical aspects of developing an intelligent network intrusion detection system that uses machine learning methods and algorithms to identify abnormal activity and unknown threats.

Existing approaches to cybersecurity and the technology stack for creating such systems are investigated, in particular, the use of Python, Scikit-learn, TensorFlow/Keras, Pandas libraries to implement traffic analysis functionality.

The novelty of the work lies in the application of a deep autoencoder architecture to implement a single-class learning strategy based on the CIC-IDS 2017 dataset.

Keywords:

Cybersecurity, Computer Network Protection, Intrusion Detection System (NIDS), Machine Learning, Artificial Intelligence, Deep Learning, Autoencoder, Isolation Forest, Unsupervised Learning, Anomaly Detection, Network Traffic Analysis, TensorFlow, Keras, Python.

Зміст

ВСТУП

У сучасному світі, де інформаційні технології стали невід`ємною частиною суспільного та економічного життя, питання безпеки комп`ютерних мереж набуває особливого значення. Розвиток цифрових технологій супроводжується зростанням кіберзагроз, серед яких віруси, атаки на відмову в обслуговуванні (DDoS), несанкціонований доступ, фішинг та експлуатація вразливостей програмного забезпечення.

Традиційні системи захисту, такі як міжмережеві екрани та класичні системи виявлення вторгнень (IDS), здебільшого базуються на сигнатурному методі аналізу. Хоча вони ефективні проти відомих загроз, вони виявляються безсилими перед новими, раніше невідомими атаками, для яких ще не створено сигнатур. Це створює критичну вразливість у периметрі безпеки організацій.

Вирішенням цієї проблеми є впровадження інтелектуальних систем, здатних аналізувати поведінку мережі та виявляти аномалії - відхилення від нормального профілю функціонування. Найбільш перспективним інструментом для цього є методи машинного навчання (Machine Learning) та штучного інтелекту. Зокрема, підходи навчання без учителя (Unsupervised Learning) та стратегія "One-Class Learning" дозволяють навчити систему розпізнавати нормальний трафік і автоматично блокувати будь-яку підозрілу активність, незалежно від її природи.

Таким чином, розробка та дослідження методів і систем захисту комп’ютерних мереж на основі машинного навчання є актуальним науково-прикладним завданням, що дозволяє підвищити рівень захищеності інформаційних інфраструктур від сучасних динамічних загроз.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ МЕТОДІВ ЗАХИСТУ КОМПЮ`ТЕРНИХ МЕРЕЖ

1.1 Актуальність захисту КМ

Комп`ютерні мережі є основою сучасної цифрової інфраструктури, забезпечуючи обмін інформацією між пристроями та користувачами. Вони можуть бути локальними (LAN), глобальними (WAN), бездротовими (WLAN) або змішаними, залежно від масштабу та призначення. Локальні мережі використовуються в організаціях для внутрішнього обміну даними, тоді як глобальні забезпечують зв`язок між віддаленими мережами та серверами.

Основним завданням комп`ютерної мережі є надійна передача даних, що передбачає використання різних технологій та протоколів. Базові компоненти мережі включають маршрутизатори, комутатори, сервери, клієнтські пристрої та мережеве програмне забезпечення. Дані передаються за допомогою протоколів, таких як TCP/IP, які визначають правила маршрутизації та обміну інформацією між вузлами мережі.

Важливою складовою функціонування мереж є захист від загроз. Згідно з аналітичним звітом у 2023 році кількість зареєстрованих та опрацьованих кіберінцидентів в Україні склала 2543 випадки. Це свідчить про те, що інтенсивність атак залишається стабільно високою, хоча і дещо знизилася порівняно з піковими показниками початку повномасштабного вторгнення. Серед зареєстрованих інцидентів критичний рівень небезпеки мали 362 події. Найбільше атак було спрямовано на урядові інституції, сили оборони, енергетику та телекомунікації.

Методи захисту комп`ютерних мереж поділяються на апаратні та програмні. Апаратні засоби, такі як міжмережеві екрани та системи виявлення

вторгнень, аналізують вхідний і вихідний трафік, блокуючи потенційно небезпечні запити. Програмні засоби включають антивірусне програмне забезпечення, засоби шифрування, двофакторну автентифікацію та системи управління доступом. Проте статистика 2023 року показує, що цього недостатньо. Для протидії сучасному шкідливому програмному забезпечені необхідно впровадити комплексні систем моніторингу подій інформаційної безпеки (SIEM) та засобів реагування на кінцевих точках (EDR).

Сучасні підходи до захисту мереж все активніше залучають технології штучного інтелекту (AI) та машинного навчання. Згідно з оглядом ринку кібербезпеки від IT Ukraine Association, Україна стала унікальним полігоном, де набувається перший у світі досвід сучасної кібервійни. Ключові тенденції розвитку галузі у 2023 році демонструють зростання напрямків DefenseTech та GovTech, де сектор кібербезпеки виступає драйвером для розвитку оборонних технологій та державних цифрових сервісів.

Особливу роль відіграє штучний інтелект, який використовується двояко: як засіб захисту для автоматичного виявлення аномалій у трафіку, так і як інструмент атак для генерації фішингових листів чи пошуку вразливостей.

Таким чином, комп`ютерні мережі є складними технологічними системами, які вимагають постійного вдосконалення методів захисту. Враховуючи зростаючі кіберзагрози, необхідно використовувати комплексний підхід до безпеки, що включає як технічні засоби, так і організаційні заходи та перехід від реактивного реагування до проактивного полювання на загрози [1]

Рисунок 1.1 – Діаграма виявлених кіберзагроз в Україні за 2023 рік[2]

1.2 Аналіз методів захисту КМ

Сучасні комп`ютерні мережі (КМ) є складними, розподіленими системами, які щоденно стикаються з безліччю загроз, від автоматизованих вірусних атак до цілеспрямованих дій зловмисників. Для забезпечення їх стабільної та безпечної роботи не існує єдиного універсального рішення; натомість використовується комплексний, багаторівневий підхід, що поєднує різноманітні методи та технології захисту.

Основна мета цих методів - гарантувати три ключові принципи інформаційної безпеки (відомі як “тріада CIA”):

Конфіденційність: Запобігання несанкціонованому доступу до даних (захист від “прослуховування” чи крадіжки).

Цілісність: Захист даних від несанкціонованої зміни чи знищення (гарантія того, що дані є точними та повними).

Доступність: Забезпечення того, що авторизовані користувачі мають доступ до мережевих ресурсів та даних тоді, коли їм це потрібно (захист від атак на відмову в обслуговуванні, як-от DDoS).

Усі існуючі методи захисту КМ можна умовно поділити на кілька категорій залежно від їхньої функції:

Методи запобігання (Prevention): Спрямовані на те, щоб не допустити інцидент. Вони діють як перша лінія оборони, контролюючи доступ та фільтруючи трафік.

Методи виявлення (Detection): Їхня задача - вчасно ідентифікувати атаку, яка вже триває або подолала перший рубіж захисту.

Методи реагування та відновлення (Response and Recovery): Активуються після виявлення інциденту для його локалізації, усунення наслідків та повернення мережі до нормального робочого стану.

Ефективна система безпеки поєднує методи з усіх цих категорій. Наприклад, якщо зловмиснику вдасться обійти засіб запобігання, його дії має зафіксувати засіб виявлення (система IDS), що активує механізм реагування.

1.2.1 Автентифікація та контроль доступу

Автентифікація дозволяє перевіряти особу користувача перед наданням доступу до мережевих ресурсів. Існує кілька основних методів автентифікації, серед яких найбільш поширеними є паролі, біометричні дані, апаратні токени та багатофакторна автентифікація.

Простий парольний захист є найменш надійним, оскільки паролі можуть бути скомпрометовані внаслідок атак типу brute force, phishing або витоків баз даних. Використання біометричних методів, таких як відбитки пальців або розпізнавання обличчя, підвищує рівень безпеки, оскільки ці дані унікальні для кожного користувача.

Проте біометрична автентифікація має свої обмеження, наприклад, можливість помилкового розпізнавання або зламування через відбитки, залишені на поверхні. Багатофакторна автентифікація (MFA) поєднує кілька методів підтвердження особи, що значно ускладнює несанкціонований доступ.

Наприклад, користувач вводить пароль, а потім підтверджує свою особу через код, отриманий на мобільний телефон. Хоча такий підхід забезпечує високий рівень безпеки, він може бути незручним для користувачів та створювати затримки у роботі.

Сучасні технології намагаються зробити автентифікацію менш нав`язливою, впроваджуючи методи безперервної автентифікації, які аналізують поведінку користувача, такі як патерни набору тексту чи руху курсора.[3,4]

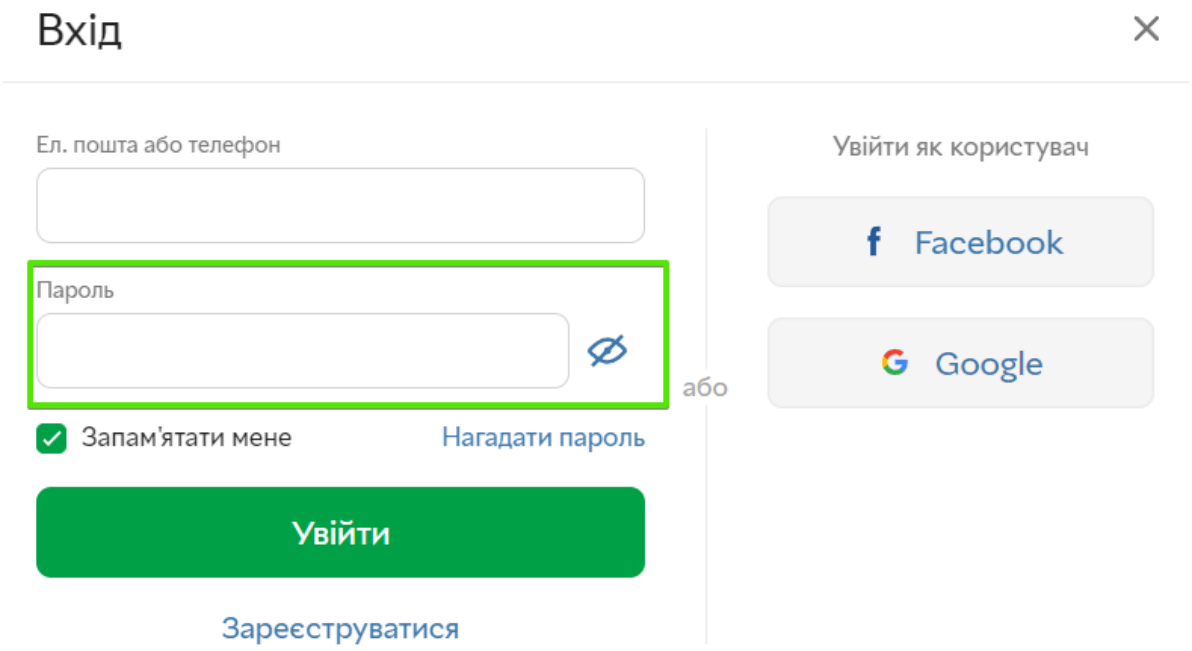

Рисунок 1.2 – Приклад автентифікації (введення паролю)

1.2.2 Криптографічні методи захисту

Криптографічні методи захисту включають в себе шифрування даних, цифрові підписи, криптографічні хеш-функції та механізми управління ключами. Шифрування даних допомагає забезпечити конфіденційність інформації, що передається через мережу.

Сучасні алгоритми шифрування, такі як AES (Advanced Encryption Standard) або RSA (Rivest-Shamir-Adleman), забезпечують високий рівень стійкості до атак. Використання SSL/TLS-протоколів є стандартним рішенням для захисту веб-трафіку, зокрема в електронній комерції та банківських операціях. Проте неправильна реалізація або використання застарілих версій цих протоколів може призвести до вразливостей, таких як атаки типу Man-in-the-Middle (MITM).

Цифрові підписи та сертифікати використовуються для перевірки справжності відправника інформації та цілісності переданих даних. Важливим аспектом криптографії є управління ключами, оскільки компрометація приватного ключа може призвести до повної втрати захисту.

Сучасні тенденції передбачають використання квантово-стійких алгоритмів, оскільки розвиток квантових обчислень може загрожувати існуючим методам криптографічного захисту. У системах безпеки криптографія застосовується для шифрування комунікацій (наприклад, TLS/SSL), електронного підпису, захисту паролів та блокчейн-технологій.

Використання криптографічних методів вимагає належного управління ключами та регулярного оновлення алгоритмів для протидії зростаючим загрозам зламу шифру.[3,5]

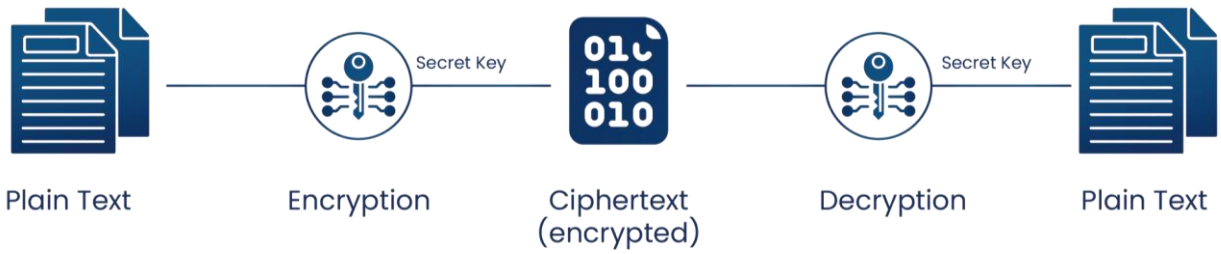

Рисунок 1.3 – Шифрування даних[6]

1.2.3 Міжмережеві екрани

Мережеві екрани (файрволи) є одним із найважливіших інструментів захисту комп`ютерних мереж, оскільки вони забезпечують контроль над мережевим трафіком на основі визначених правил і політик безпеки. Основне їхнє завдання полягає у дозволі або блокуванні певних з`єднань, що дає змогу запобігти несанкціонованому доступу та атакам ззовні.

Існують різні типи файрволів, кожен з яких має свої переваги та обмеження:

Файрволи рівня пакетів (packet-filtering firewalls) працюють на мережевому та транспортному рівнях моделі OSI. Вони аналізують кожен пакет окремо, звертаючи увагу лише на його заголовок (IP-адреси відправника та одержувача, номер порту, протокол). Їхня перевага - висока швидкість роботи та мінімальне навантаження на систему, але вони не здатні перевіряти вміст трафіку та виявляти складні загрози.

Станційні файрволи (stateful firewalls) відслідковують стан з`єднань, що дозволяє приймати рішення з урахуванням контексту мережевої сесії. Це забезпечує більш ефективний контроль порівняно з простим аналізом заголовків пакетів, проте такі системи потребують більших ресурсів.

Файрволи наступного покоління (Next-Generation Firewalls, NGFW) поєднують традиційні функції із сучасними технологіями: глибоким аналізом

трафіку (Deep Packet Inspection, DPI), виявленням та блокуванням відомих і невідомих загроз, інтеграцією з системами запобігання вторгненням (IPS), а також підтримкою шифрованого трафіку (SSL/TLS inspection). NGFW здатні контролювати додатки, розпізнавати підозрілу поведінку та здійснювати моніторинг у реальному часі.

Хмарні файрволи (Cloud Firewalls) - це відносно новий напрям, який особливо актуальний для компаній з розподіленою інфраструктурою та використанням хмарних сервісів. Вони дозволяють централізовано керувати політиками безпеки, масштабуються залежно від навантаження та інтегруються з іншими сервісами кіберзахисту.

Важливою умовою ефективності файрволів є регулярне оновлення правил та політик, оскільки сучасні кіберзагрози постійно змінюються і швидко адаптуються до нових умов. Недостатньо просто встановити файрвол - необхідно проводити аудит його налаштувань, реагувати на виявлені атаки та забезпечувати актуальність сигнатур.

Сучасні підходи до безпеки також включають Zero Trust Network Access (ZTNA) - архітектуру, що передбачає принцип «нікому не довіряти за замовчуванням». Це означає, що навіть внутрішні користувачі чи пристрої проходять автентифікацію та авторизацію для кожної нової сесії чи транзакції, що значно знижує ризик внутрішніх загроз і компрометації мережі.

Таким чином, у світі динамічного розвитку технологій та зростання складності атак, файрволи залишаються невід`ємною складовою комплексних стратегій кібербезпеки, забезпечуючи першу лінію оборони для інформаційних систем підприємств і організацій.[7]

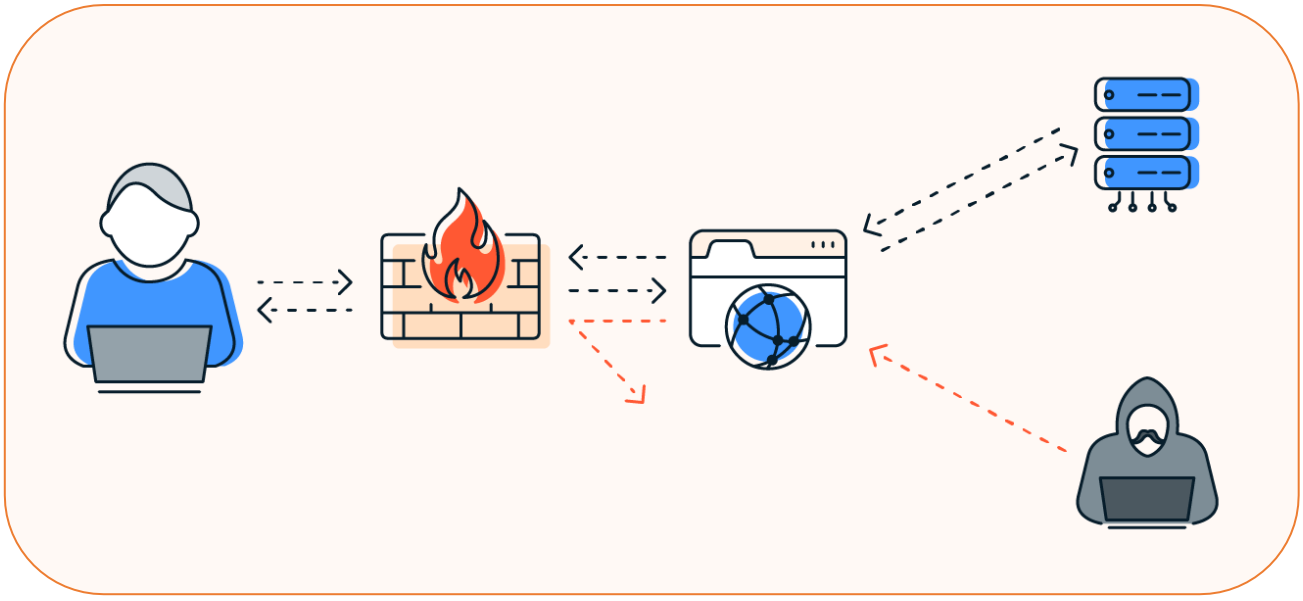

Рисунок 1.4 – Візуалізація роботи файрволу

1.2.4 Системи виявлення та запобігання вторгнень

Системи виявлення та запобігання вторгнень (IDS/IPS) дозволяють ідентифікувати та блокувати загрози в режимі реального часу. IDS (Intrusion Detection System) виконує моніторинг мережевого трафіку та аналізує його для виявлення аномалій або підозрілої активності, тоді як IPS (Intrusion Prevention System) не лише виявляє потенційні атаки, а й запобігає їм шляхом автоматичного реагування. Поєднання цих систем значно підвищує рівень безпеки, дозволяючи підприємствам та організаціям швидко реагувати на кіберзагрози.

IDS бувають двох основних типів: сигнатурні та поведінкові. Сигнатурні системи працюють на основі відомих шаблонів атак (сигнатур), що робить їх ефективними у боротьбі з уже відомими загрозами. Однак вони мають обмеження у виявленні нових або модифікованих атак, які ще не були внесені в базу даних. Поведінкові IDS аналізують мережевий трафік на предмет аномалій, що можуть вказувати на потенційні загрози. Вони використовують машинне навчання або

евристичні методи для визначення підозрілої активності, що дозволяє їм виявляти нові види атак, проте це може призводити до помилкових спрацьовувань.

IPS працюють у більш проактивному режимі та можуть автоматично блокувати шкідливий трафік або підозрілі дії. Наприклад, якщо система виявляє несанкціоновані спроби отримати доступ до корпоративної мережі, вона може автоматично заблокувати IP-адресу порушника або закрити вразливий порт. Сучасні IPS використовують методи аналізу поведінки та штучного інтелекту для швидкого прийняття рішень, що значно зменшує ймовірність успішної атаки.

Поєднання IDS та IPS забезпечує комплексний підхід до захисту мереж, дозволяючи не лише виявляти атаки, але й ефективно запобігати їм. Багато сучасних рішень використовують гібридний підхід, комбінуючи сигнатурний аналіз із поведінковим, що дає змогу досягти максимальної ефективності. Такі системи активно застосовуються у фінансових установах, дата-центрах, урядових організаціях та великих корпораціях, де необхідний високий рівень безпеки.

Одним із важливих напрямів розвитку IDS/IPS є інтеграція з SIEM (Security Information and Event Management) для збору, аналізу та кореляції подій безпеки в реальному часі. Це дозволяє не лише локально реагувати на загрози, а й будувати глобальну картину загроз для всієї організації. Використання штучного інтелекту та машинного навчання в IDS/IPS сприяє зниженню кількості помилкових спрацьовувань, а також підвищенню ефективності виявлення складних атак, таких як APT (Advanced Persistent Threat).

Таким чином, сучасні системи виявлення та запобігання вторгнень є важливими складовими стратегії кібербезпеки. Вони не лише забезпечують моніторинг та аналіз загроз, а й активно блокують атаки, мінімізуючи ризики для інформаційних систем. Зі зростанням складності кібератак роль IDS/IPS продовжує зростати, що вимагає подальшого розвитку технологій у цьому напрямі.[8,9]

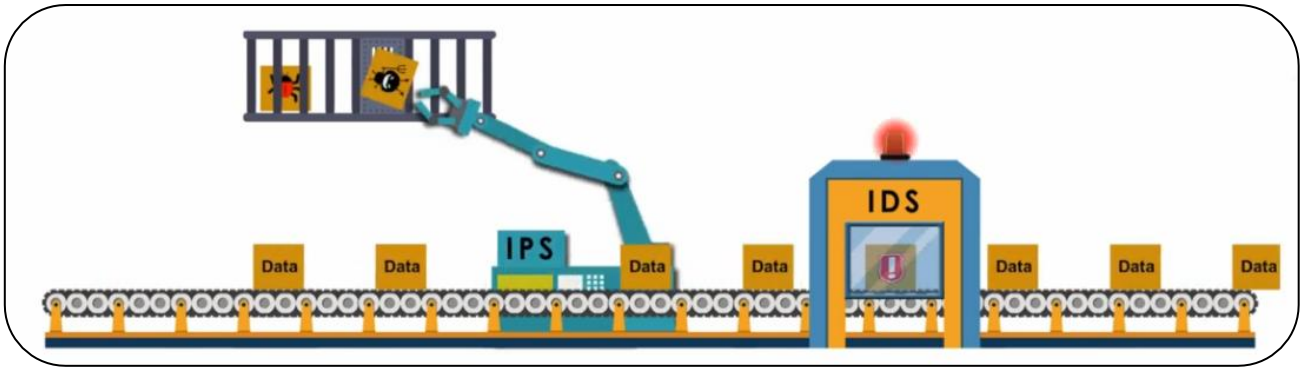

Рисунок 1.5 – Система IPS та IDS

1.2.5 Контроль доступу

В умовах стрімкого зростання кількості кіберзагроз та витоку конфіденційної інформації контроль доступу набуває критично важливого значення. Існують різні моделі контролю доступу, серед яких найбільш поширеними є дискреційний (DAC), мандатний (MAC) та рольовий (RBAC) підходи.

Дискреційний контроль передбачає, що власник об`єкта самостійно визначає, хто може отримати до нього доступ. Хоча цей метод забезпечує гнучкість, він є вразливим до атак соціальної інженерії та зловживань правами.

Мандатний контроль, навпаки, базується на жорстких політиках безпеки та класифікації даних, що дозволяє запобігти несанкціонованому доступу навіть адміністраторам.

У сучасних системах широко використовується рольовий контроль доступу, який спрощує управління великими мережами, призначаючи користувачам певні ролі з визначеними привілеями. Незалежно від обраної моделі, важливим є регулярний аудит та оновлення політик доступу, оскільки зміни в організаційній структурі можуть призвести до потенційних загроз безпеці.

Для підвищення ефективності контролю доступу часто використовують багатофакторну автентифікацію та адаптивні механізми управління ідентифікацією користувачів. Використання сучасних інструментів контролю доступу, таких як системи управління ідентифікацією та доступом (IAM), дозволяє централізовано контролювати права доступу та знижувати ризик витоку даних.

В умовах цифрової трансформації контроль доступу стає не лише інструментом безпеки, а й частиною стратегії захисту критичних інформаційних активів компаній та організацій.[10]

1.2.6 Антивірусні та антимальварні системи

Зі збільшенням кількості кіберзагроз та появою нових методів атак важливість використання антивірусного захисту зростає. Сучасні антивірусні рішення працюють на основі сигнатурного аналізу, евристичних методів та поведінкових алгоритмів для виявлення загроз.

Сигнатурний аналіз порівнює файли та процеси з базою відомих шкідливих програм, що дозволяє швидко виявити вже відомі віруси. Проте цей метод неефективний проти нових загроз, які ще не внесені до бази даних.

Евристичний аналіз розпізнає потенційно небезпечні програми на основі їхньої структури та поведінки, що дозволяє ідентифікувати невідомі віруси. Поведінковий аналіз здійснює моніторинг програм у режимі реального часу та блокує їхню діяльність у разі виявлення підозрілих дій. Сучасні антивірусні рішення часто поєднують кілька методів для підвищення ефективності виявлення загроз.

Важливим аспектом є регулярне оновлення антивірусних баз та програмного забезпечення, оскільки кіберзлочинці постійно розробляють нові

способи обходу захисту. Крім того, значну роль відіграє освітня робота серед користувачів, оскільки більшість атак здійснюється через соціальну інженерію.

Використання антивірусних рішень у поєднанні з проактивними методами захисту, такими як система виявлення вторгнень (IDS) та аналіз поведінки користувачів (UEBA), дозволяє суттєво знизити ризик зараження та втрати даних. У корпоративному середовищі антивірусні рішення часто інтегруються з іншими системами інформаційної безпеки для створення комплексного підходу до кіберзахисту.[11]

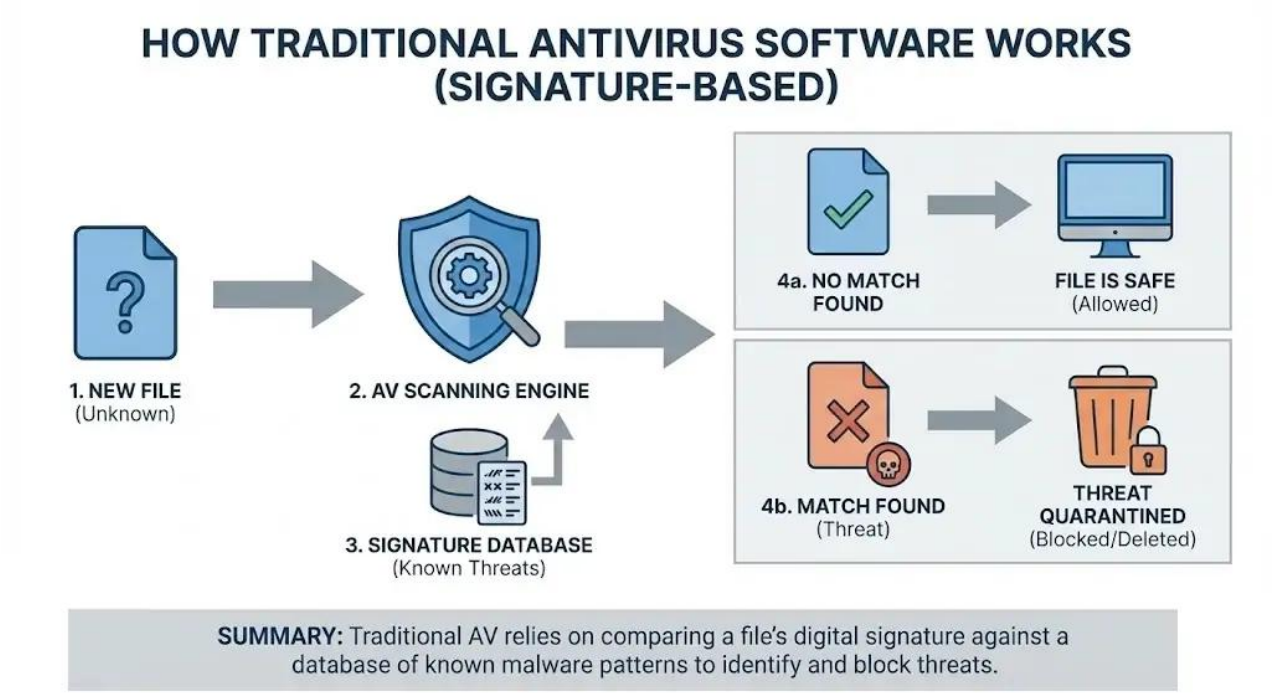

Рисунок 1.6 – Робота антивірусної програми[12]

1.2.7 Захист бездротових мереж

Безпека бездротових мереж є критично важливим елементом захисту комп`ютерних систем, адже саме Wi-Fi найчастіше використовується як точка входу для зловмисників. Однією з ключових проблем є несанкціонований доступ

до мережі, який може бути здійснений через слабкі або стандартні паролі, а також відкриті точки доступу, що залишаються без захисту.

Використання сучасних протоколів шифрування, таких як WPA3, значно підвищує рівень безпеки, адже він забезпечує стійкість до словникових атак і захищає від перехоплення трафіку навіть у разі слабких паролів. Проте навіть мережі з активованим WPA3 можуть бути вразливими: наприклад, зловмисники можуть застосувати атаку Evil Twin, створюючи підроблену точку доступу, яка імітує легітимну мережу, або ініціювати атаки типу Denial of Service (DoS/DDoS), що призводять до відмови в обслуговуванні користувачів.

Серед додаткових методів захисту бездротових мереж застосовуються:

Фільтрація MAC-адрес - дозволяє обмежити доступ лише для певних пристроїв, проте може бути обійдена через підміну MAC-адреси.

Приховування SSID - робить мережу менш помітною для сторонніх користувачів, але не є абсолютним захистом.

Багатофакторна автентифікація (MFA) – додає додатковий рівень безпеки, вимагаючи, крім пароля, ще один фактор (наприклад, SMS-код, біометричні дані або апаратний токен).

Сегментація мережі (VLAN) – дозволяє відокремлювати корпоративний трафік від гостьового, що мінімізує ризики поширення загроз у мережі.

Важливим аспектом є також правильне налаштування обладнання: необхідно вимикати застарілі протоколи безпеки (WEP, WPA), використовувати складні паролі адміністратора та застосовувати ізоляцію клієнтів у Wi-Fi для запобігання їхньому прямому доступу один до одного.

Крім того, ефективність захисту залежить від регулярного оновлення мікропрограмного забезпечення (firmware) роутерів і точок доступу, оскільки виробники постійно виправляють вразливості.

У корпоративному середовищі дедалі більшого поширення набувають розподілені системи управління бездротовими мережами (Wireless LAN

Controllers, WLC). Вони забезпечують централізоване керування десятками й сотнями точок доступу, дозволяють здійснювати моніторинг у режимі реального часу, застосовувати політики безпеки для різних сегментів мережі та оперативно реагувати на підозрілу активність. У поєднанні з системами WIDS/WIPS (Wireless Intrusion Detection/Prevention Systems) це дає змогу виявляти атаки на бездротову інфраструктуру та блокувати їх на ранніх етапах.

Таким чином, комплексний підхід до захисту Wi-Fi включає використання сучасних протоколів шифрування, багаторівневих методів автентифікації, централізованого управління й моніторингу, а також регулярне оновлення обладнання. Тільки за таких умов можна гарантувати високий рівень безпеки бездротових мереж у сучасному цифровому середовищі. [13]

1.2.8 Захист від DDoS-атак

DDoS-атаки (Distributed Denial of Service) є одним із найпоширеніших типів кіберзагроз, які можуть призвести до тимчасової або повної недоступності веб-сайтів, серверів та інших мережевих ресурсів. Метою таких атак є перевантаження інфраструктури жертви великою кількістю запитів, що унеможливлює нормальну роботу системи. Для ефективного захисту від DDoS-атак використовуються різні методи та технології, включаючи хмарні рішення, апаратні засоби та алгоритми штучного інтелекту.

Одним із ключових способів захисту є використання систем виявлення та запобігання вторгнень (IDS/IPS), які дозволяють ідентифікувати та блокувати підозрілу активність у мережі. Такі системи аналізують вхідний трафік і можуть автоматично фільтрувати аномальні запити, тим самим зменшуючи навантаження на сервер. Наприклад, IPS можуть автоматично блокувати IP-адреси, з яких надходить підозріло велика кількість запитів за короткий проміжок часу.

Ще одним важливим методом є балансування навантаження (Load Balancing), яке дозволяє рівномірно розподіляти трафік між кількома серверами, запобігаючи перевантаженню одного вузла. Це особливо ефективно для веб-сайтів із великою кількістю відвідувачів, оскільки навантаження може динамічно змінюватися залежно від рівня атак. Балансувальники навантаження можуть працювати на рівні мережевого трафіку (L4) або на рівні додатків (L7), що дає змогу більш точно аналізувати запити.

Хмарні сервіси захисту від DDoS-атак, такі як Cloudflare, Akamai та AWS Shield, пропонують спеціалізовані рішення для фільтрації шкідливого трафіку на рівні глобальної інфраструктури. Вони використовують розподілені мережі серверів, які можуть поглинати надмірний трафік, не допускаючи його до кінцевої точки. Фільтрація трафіку на рівні брандмауерів (Firewalls) та систем глибокого аналізу пакетів (DPI) також є ефективним методом боротьби з DDoS-атаками. Вони дозволяють блокувати трафік на основі його характерних ознак, наприклад, визначати атаки SYN Flood або UDP Flood. Деякі сучасні брандмауери використовують інтеграцію зі SIEM-системами для централізованого збору та аналізу інформації про загрози.

Одним із новітніх підходів є використання технологій штучного інтелекту та поведінкового аналізу, які дозволяють автоматично визначати аномальну активність та швидко реагувати на загрози. Наприклад, нейромережі можуть аналізувати типові шаблони поведінки користувачів і автоматично блокувати підозрілу активність. Такий підхід дозволяє значно знизити рівень помилкових спрацьовувань і забезпечує гнучке налаштування системи безпеки.

Захист від DDoS-атак є критично важливим для компаній, які працюють у фінансовій сфері, електронній комерції та інших галузях, що залежать від безперебійної роботи веб-ресурсів. Використання комбінації технологій, таких як балансування навантаження, хмарні сервіси, глибока фільтрація трафіку та поведінковий аналіз, дозволяє мінімізувати ризики та забезпечити стабільну

роботу інформаційних систем. Оскільки методи DDoS-атак постійно розвиваються, ефективний захист вимагає постійного оновлення механізмів безпеки та адаптації до нових загроз.[14]

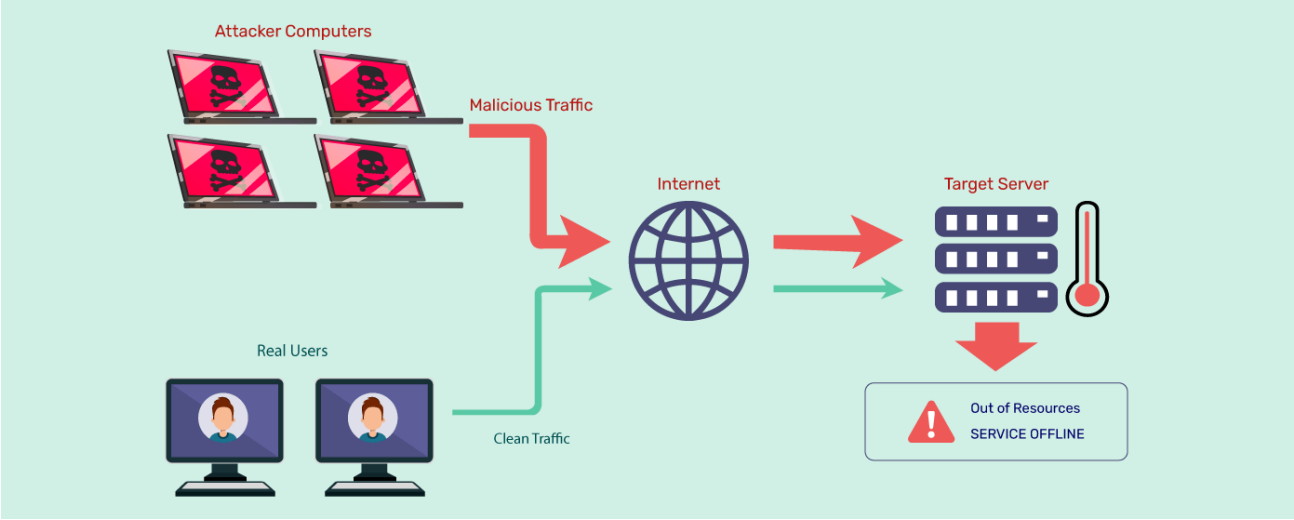

Рисунок 1.7 – DDOS атака[15]

1.2.9 Резервне копіювання та аварійне відновлення

Резервне копіювання та аварійне відновлення забезпечує безперервність бізнесу та захисту даних у разі непередбачуваних подій. Втрата критично важливої інформації через збої обладнання, кібератаки, людські помилки або стихійні лиха може призвести до серйозних фінансових та репутаційних втрат. Саме тому організації впроваджують комплексні стратегії резервного копіювання та аварійного відновлення, щоб гарантувати збереження даних і швидке відновлення функціональності систем.

Одним із найважливіших аспектів резервного копіювання є вибір відповідної стратегії збереження даних. Найпоширенішими методами є повне, диференційне та інкрементне резервне копіювання. Повне резервне копіювання передбачає створення повної копії всіх даних, що забезпечує високу надійність, але потребує значного обсягу дискового простору. Диференційне резервне

копіювання зберігає лише зміни, зроблені після останнього повного копіювання, що дозволяє зменшити навантаження на систему. Інкрементне копіювання є ще більш ефективним, оскільки зберігає тільки нові або змінені файли, що значно знижує витрати на збереження даних.

Сучасні системи резервного копіювання часто використовують хмарні технології, які забезпечують високу доступність і гнучкість у збереженні даних. Хмарні рішення дозволяють автоматизувати процес резервного копіювання, забезпечуючи безпечне збереження інформації на віддалених серверах. Це особливо важливо у разі катастрофічних подій, коли локальні резервні копії можуть бути втрачені разом з основною інфраструктурою. Крім того, хмарні сервіси пропонують розширені можливості шифрування та автентифікації, що підвищує рівень безпеки збережених даних.

Аварійне відновлення передбачає набір процедур та технологій, які дозволяють швидко відновити працездатність інформаційних систем після збою. Важливим елементом цього процесу є план аварійного відновлення, який включає перелік критично важливих систем, інструкції для їх відновлення та контакти відповідальних осіб. Добре розроблений план дозволяє мінімізувати час простою та зменшити втрати, пов`язані з інцидентами.

Одним із підходів до аварійного відновлення є використання технологій реплікації даних. Реплікація дозволяє створювати точні копії інформації у реальному часі, що гарантує її доступність навіть у разі виходу з ладу основної системи. Реплікація може здійснюватися як у локальному середовищі, так і у віддалених дата-центрах, що забезпечує додатковий рівень захисту.

Автоматизація процесів резервного копіювання та відновлення є ще одним важливим фактором підвищення ефективності. Використання спеціалізованого програмного забезпечення дозволяє налаштувати регулярне резервне

копіювання, контроль за його успішністю та автоматизоване відновлення у разі потреби. Такий підхід значно знижує ймовірність людських помилок і забезпечує високу швидкість відновлення даних.

Тестування резервних копій та процедур аварійного відновлення є необхідною складовою безпекової стратегії будь-якої організації. Регулярні перевірки дозволяють виявити потенційні проблеми у процесах збереження та відновлення даних, а також забезпечити їхню ефективність у реальних умовах. Без належного тестування навіть найкращі системи резервного копіювання можуть виявитися неефективними у критичний момент.

У сучасних умовах кіберзагроз резервне копіювання також використовується як засіб боротьби з програмами-вимагачами (ransomware). Ці шкідливі програми шифрують файли та вимагають викуп за їх розшифрування. Маючи актуальні резервні копії, організації можуть швидко відновити свої дані без необхідності сплачувати кіберзлочинцям.[16]

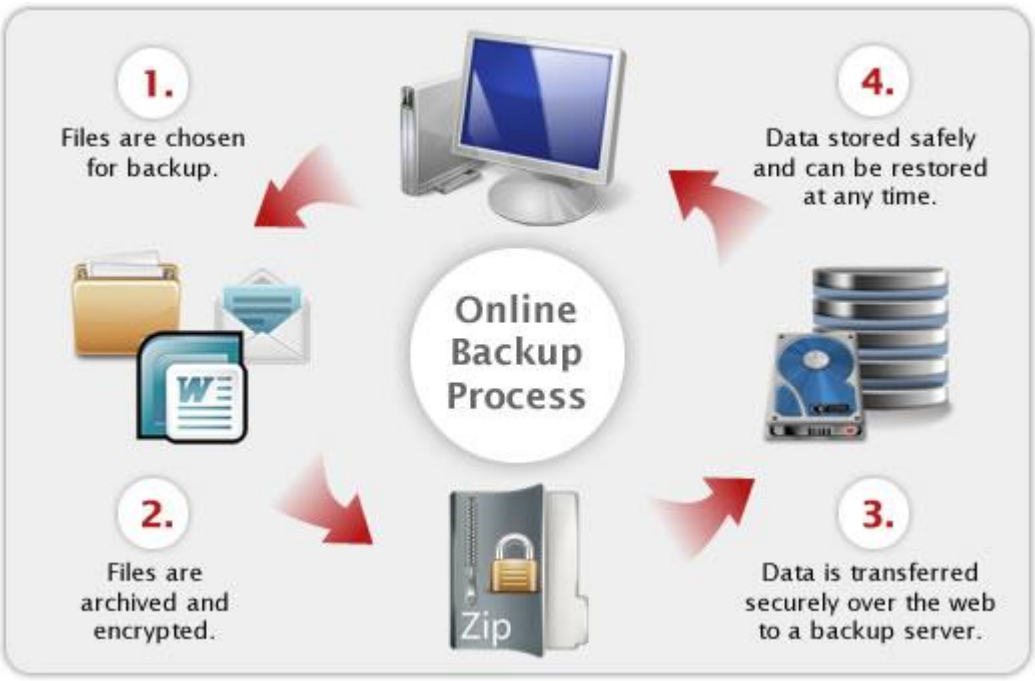

Рисунок 1.8 – Резервне копіювання та відновлення[17]

1.3 Патентний пошук

Аналіз патентів у сфері кібербезпеки є важливим етапом досліджень, що дозволяє виявити сучасні розробки та технологічні рішення у цій галузі. Нижче наведено аналіз кількох патентів, що охоплюють різні аспекти захисту комп`ютерних мереж, зокрема методи шифрування, виявлення загроз, управління доступом та інші інноваційні підходи.

1.3.1 Система динамічного контролю доступу

Цей патент описує інноваційну систему динамічного контролю доступу, яка базується на поведінковому аналізі користувачів та адаптивному реагуванні на потенційні загрози. На відміну від традиційних методів, ця технологія дозволяє автоматично змінювати рівень доступу на основі оцінки ризиків. У ній використовується машинне навчання для визначення нетипових дій користувачів та блокування потенційних атак. Основні переваги цієї системи включають високу точність виявлення загроз, автоматичну адаптацію до нових ризиків та мінімізацію впливу людського фактора. Дослідження показують, що використання такого підходу дозволяє скоротити кількість успішних атак на корпоративні мережі на 40%. Крім того, система може інтегруватися з існуючими засобами безпеки, що спрощує її впровадження у великих організаціях.

Система динамічного контролю доступу працює за принципом багаторівневої перевірки користувачів. Вона враховує такі фактори, як місцезнаходження, історію входів, рівень ризику підключеного пристрою та навіть поведінкові особливості під час використання корпоративної мережі. Наприклад, якщо користувач входить у систему з нового пристрою або з

незвичного місцезнаходження, система може вимагати додаткову автентифікацію або тимчасово обмежити доступ. Це значно зменшує ймовірність атак, заснованих на компрометації облікових даних.

Окрім основного функціоналу, система має можливість інтеграції з SIEM-системами (Security Information and Event Management) для централізованого збору та аналізу подій безпеки. Це забезпечує всебічний контроль за всіма аспектами безпеки корпоративного середовища. За рахунок використання машинного навчання система постійно оновлюється та вдосконалюється, що дозволяє їй ефективно протидіяти новим типам атак. Дослідження показали, що впровадження цієї технології у великих організаціях призводить до зниження витрат на усунення наслідків кіберзагроз на 30%.[18]

1.3.2 Метод багатофакторної автентифікації на основі біометричних даних

У цьому патенті представлено метод багатофакторної автентифікації, що використовує комбінацію біометричних даних, зокрема відбитків пальців, розпізнавання обличчя та голосовий аналіз. На відміну від класичних підходів, які ґрунтуються лише на одному факторі перевірки (наприклад, паролі чи смарт-карти), запропонована технологія поєднує кілька незалежних рівнів ідентифікації. Це значно ускладнює несанкціонований доступ і знижує ймовірність компрометації облікового запису.

Важливою особливістю даного методу є використання штучних нейронних мереж для аналізу біометричних параметрів. Завдяки цьому досягається висока точність розпізнавання користувача, зменшується кількість помилкових спрацьовувань (false positives) та відмов у доступі (false negatives). Система може самонавчатися, підвищуючи ефективність роботи з часом, що особливо

актуально у випадках, коли біометричні характеристики користувача частково змінюються (наприклад, зміна голосу, незначні травми пальців чи зміни зовнішності).

Практичне застосування такого методу є надзвичайно широким. У корпоративних мережах він дає змогу підвищити рівень захисту конфіденційної інформації та критично важливих ресурсів. У банківських системах та платіжних сервісах біометрична автентифікація мінімізує ризик шахрайських дій, забезпечуючи високий рівень довіри з боку клієнтів. Крім того, метод може застосовуватися у сфері електронного урядування, охорони здоров`я (для захисту електронних медичних записів) та у мобільних додатках, де необхідна швидка і надійна перевірка користувача.

Ще однією перевагою є зручність використання: біометричні дані не можна «забути» чи «загубити», на відміну від традиційних паролів чи фізичних токенів. Сучасні реалізації системи демонструють швидкість обробки запитів менше 1 секунди, що робить технологію придатною для щоденного використання навіть у високонавантажених системах.[19]

1.3.3 Технологія виявлення загроз на основі аналізу поведінки мережевого трафіку

Патент стосується новітнього підходу до виявлення мережевих загроз, заснованого на поведінковому аналізі трафіку. На відміну від класичних сигнатурних систем, які орієнтуються на відомі шаблони атак, ця технологія дозволяє виявляти невідомі або модифіковані загрози. Використання методів штучного інтелекту та машинного навчання робить можливим аналіз великих обсягів мережевих даних у реальному часі.

Система відслідковує аномальні патерни у поведінці користувачів, серверів і пристроїв, які можуть свідчити про потенційні атаки, наприклад:

- DDoS-атаки – різке зростання трафіку з однієї або кількох адрес;
- SQL-ін`єкції – нетипові запити до баз даних;
- несанкціонований доступ – підозріла активність із нових географічних зон чи пристроїв.

Основною перевагою цієї технології є її адаптивність: система здатна автоматично навчатися новим сценаріям атак і блокувати загрози в реальному часі, зменшуючи залежність від людського втручання. Дослідження показують, що впровадження поведінкового аналізу дозволяє знизити кількість невиявлених атак на 35%, що суттєво підвищує загальну стійкість корпоративних мереж до сучасних кіберзагроз. [20]

1.3.4 Методологія хмарних файрволів нового покоління

Розглянутий патент описує архітектуру хмарних файрволів нового покоління (Cloud NGFW), які здатні забезпечити гнучкий та масштабований захист як для локальних мереж, так і для розподілених хмарних середовищ. На відміну від класичних апаратних файрволів, хмарні рішення інтегруються безпосередньо у мережеву інфраструктуру постачальників послуг, дозволяючи захищати трафік незалежно від фізичного розташування користувача чи сервера.

Використання штучного інтелекту та аналізу великих даних (Big Data Analytics) дозволяє значно підвищити точність виявлення загроз і мінімізувати помилкові спрацьовування. Особливістю цієї методики є можливість централізованого управління політиками безпеки у глобальному масштабі: адміністратор може в єдиній консолі налаштовувати правила доступу для всіх

офісів та філій компанії, що критично важливо для міжнародних корпорацій і провайдерів хмарних сервісів.

Порівняно з традиційними підходами, хмарні файрволи дозволяють зменшити час виявлення та блокування загроз на 50%, забезпечуючи безперервний моніторинг і динамічне оновлення правил у режимі реального часу. Це робить їх ключовим елементом сучасних Zero Trust-архітектур, де довіра до будь-якого трафіку завжди перевіряється. [21]

1.3.5 Автоматизована система моніторингу та запобігання кібератакам

Даний патент описує інтелектуальну систему кіберзахисту, яка здатна не лише виявляти атаки під час їх виконання, але й прогнозувати їх ще до фактичного здійснення. Система використовує машинне навчання та аналіз великих даних, що дозволяє розпізнавати навіть мінімальні відхилення у поведінці користувачів, серверів і мережевого трафіку.

Основною перевагою є здатність системи до самонавчання – вона автоматично вдосконалює алгоритми виявлення нових загроз, без необхідності ручного оновлення сигнатур чи правил. Такий підхід особливо ефективний у сфері фінансових установ, державних організацій та медичних закладів, де рівень ризику кіберзагроз є надзвичайно високим.

Технічні характеристики рішення вражають: система здатна обробляти до 10 мільйонів пакетів за секунду, що гарантує безперервний моніторинг та захист у режимі real-time навіть у високонавантажених мережах. Додатково система підтримує інтеграцію з SIEM та SOAR-платформами, що дозволяє автоматично реагувати на інциденти та запускати сценарії нейтралізації атак без участі людини.[22]

Висновки до розділу:

У першому розділі магістерської роботи проведено комплексний теоретико-аналітичний огляд сучасного стану кібербезпеки та методів захисту комп'ютерних мереж. В результаті аналізу базових архітектур встановлено, що стек протоколів TCP/IP містить фундаментальні вразливості на різних рівнях моделі OSI, які в умовах розмивання традиційного мережевого периметра та впровадження хмарних технологій створюють численні вектори для потенційних атак. Дослідження динаміки кіберзагроз засвідчило еволюцію методів зловмисників від масових розсилок шкідливого ПЗ до складних цілеспрямованих атак (APT) та розподілених атак на відмову в обслуговуванні (DDoS) прикладного рівня, які характеризуються високою скритністю та здатністю обходити стандартні механізми фільтрації.

Критичний аналіз існуючих систем захисту, зокрема міжмережевих екранів та сигнатурних систем виявлення вторгнень (IDS), виявив суттєві обмеження їхньої ефективності. Головним недоліком традиційних підходів визначено їхню реактивну природу та залежність від баз відомих сигнатур, що робить систему беззахисною перед загрозами «нульового дня», поліморфними вірусами та атаками, прихованими в шифрованому трафіку.

На основі порівняльного аналізу методів детектування зроблено висновок, що найбільш перспективним шляхом подолання цих обмежень є перехід до інтелектуальних систем аналізу поведінки. Обґрунтовано, що застосування методів машинного навчання, особливо алгоритмів навчання без учителя, дозволяє виявляти аномалії в мережевому трафіку без попереднього знання про структуру атаки, що підтверджує актуальність та доцільність розробки власної моделі інтелектуальної системи виявлення мережевих аномалій (ML-NADS) у рамках даного дослідження.

РОЗДІЛ 2. ПРОЄКТУВАННЯ МОДЕЛІ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ВИЯВЛЕННЯ ВТОРГНЕНЬ

2.1 Вибір середовища розробки та мови програмування

Вибір інструментальних засобів для реалізації системи виявлення мережевих аномалій базувався на комплексному аналізі вимог до продуктивності розробки, швидкодії обчислень та наявності спеціалізованих бібліотек. В якості основної мови програмування було обрано Python (версія 3.11). На сьогоднішній день Python є домінуючим інструментом у сферах науки про дані, штучного інтелекту та інформаційної безпеки.

Python - це високорівнева інтерпретована мова програмування загального призначення. Вона дозволяє розробнику використовувати об`єктно-орієнтований, процедурний та функціональний стилі програмування в межах одного проєкту. Така гнучкість є критично важливою при створенні складних систем, де модуль збору даних може бути реалізований процедурно для забезпечення швидкодії, а архітектура нейронних мереж описується в об`єктно-орієнтованому стилі.

У контексті даної магістерської роботи, де основна увага приділяється експериментам з архітектурами автокодувальників (Autoencoder) та налаштуванню алгоритмів ізоляції (Isolation Forest), використання Python дозволило забезпечити швидку ітеративність процесу розробки та тестування гіпотез.

Важливим аспектом вибору Python є його архітектура управління пам`яттю та виконання коду. Python використовує автоматичне керування пам`яттю за допомогою вбудованого збирача сміття (Garbage Collector). Механізм базується

на підрахунку посилань (reference counting) та алгоритмі виявлення циклічних посилань. Для системи NIDS, яка працює з великими масивами мережевих даних (датасет CIC-IDS 2017 містить мільйони записів), це означає, що розробнику не потрібно вручну виділяти та звільняти пам`ять, як це робиться в мовах C або C++. Це мінімізує ризики витоку пам`яті (memory leaks), які могли б призвести до нестабільності системи під час тривалого моніторингу мережі.

Однією з особливостей стандартної реалізації Python (CPython) є наявність глобального блокування інтерпретатора (Global Interpreter Lock - GIL). GIL - це м`ютекс, який запобігає одночасному виконанню байт-коду Python кількома потоками. Хоча це часто вважається недоліком для багатопотокових обчислень, у контексті даної роботи це не стало обмеженням. Основне обчислювальне навантаження в системі ML-NADS припадає на матричні операції та навчання нейронних мереж. Бібліотеки, що використовуються для цих задач (NumPy, TensorFlow), реалізовані на низькорівневих мовах C/C++ і здатні “відпускати” GIL під час виконання інтенсивних обчислень. Таким чином, Python виступає ефективним “клеєм” (glue language), що керує високопродуктивними обчислювальними ядрами, написаними на компільованих мовах, забезпечуючи продуктивність, близьку до нативної.

Використання версії Python 3.11 у даній роботі також є стратегічним рішенням. Ця версія включає значні оптимізації в рамках проєкту “Faster CPython”, зокрема адаптивний інтерпретатор, який спеціалізується на виконанні часто повторюваного коду. Для системи реального часу, яка циклічно обробляє вхідні мережеві пакети, це забезпечує зниження затримок (latency) та підвищення загальної пропускної здатності аналізатора.[23]

Для створення програмного комплексу було обрано інтегроване середовище розробки Visual Studio Code. Це сучасний, кросплатформний

редактор вихідного коду, розроблений компанією Microsoft, який поєднує в собі простоту текстового редактора з потужними функціями повноцінної IDE.

Вибір VS Code обумовлений специфікою проєкту, яка передбачає “гібридну” розробку: поєднання дослідницької роботи в ноутбуках (Jupyter Notebooks) та написання продуктового коду (Python scripts). Традиційні підходи часто вимагають використання двох різних інструментів, наприклад Jupyter Lab для аналізу даних та PyCharm для написання скриптів, що ускладнює робочий процес. VS Code дозволяє об`єднати ці етапи в єдиному інтерфейсі.

VS Code побудований на базі фреймворку Electron, що дозволяє йому бути легковаговим порівняно з “важкими” IDE, такими як PyCharm або Visual Studio. Для магістерської роботи, яка виконується на персональному комп`ютері, оптимізація використання оперативної пам`яті (RAM) є критичною, особливо коли паралельно з середовищем розробки запущено процес навчання нейронної мережі, що споживає значні ресурси.

Архітектура VS Code базується на модульній системі. “Ядро” редактора містить лише базові функції редагування тексту та управління файлами. Весь спеціалізований функціонал (підтримка Python, робота з Git, запуск Jupyter) додається через систему розширень. Це дозволило налаштувати середовище виключно під потреби проєкту ML-NADS, не перевантажуючи його зайвими інструментами для веб-розробки чи інших мов.

Для забезпечення ефективної роботи з кодом системи було встановлено та налаштовано ряд ключових розширень, які перетворюють VS Code на потужну станцію для Data Science.

1. Python Extension

Це базове розширення, яке забезпечує інтеграцію інтерпретатора Python з редактором. У рамках роботи воно виконувало такі функції:

- Управління віртуальними середовищами (venv): Проєкт використовує ізольоване середовище для бібліотеки TensorFlow. Розширення автоматично детектує віртуальне середовище та налаштовує шляхи (PATH), що гарантує коректний імпорт бібліотек без конфліктів із системним Python.
- Linting (Аналіз коду): Автоматична перевірка синтаксису та стилю коду (PEP 8) в реальному часі. Це дозволило уникнути багатьох синтаксичних помилок ще на етапі написання коду модулів flow_extractor.py та main.py.

2. Pylance

Це високопродуктивний мовний сервер (Language Server), який працює поверх розширення Python. Pylance забезпечує функціонал IntelliSense - розумне автодоповнення коду.

У проєкті це було особливо корисно при роботі з бібліотекою TensorFlow. Наприклад, при написанні model.fit(), Pylance автоматично підказував список доступних аргументів (epochs, batch_size, callbacks), що значно прискорило процес написання коду та зменшило необхідність постійного звернення до зовнішньої документації.

3. Jupyter Extension

Це розширення є критично важливим для даної роботи. Воно дозволяє відкривати, редагувати та запускати файли .ipynb безпосередньо у VS Code.

Завдяки цьому було реалізовано безшовний перехід між етапами:

1. Навчання моделі у program.ipynb з візуалізацією графіків.
2. Миттєве перенесення успішних фрагментів коду у файл main.py для фінальної реалізації.

Можливість бачити графіки навчання та таблиці прямо поруч із кодом скрипта значно підвищила ергономіку роботи.

Розробка складних алгоритмів, неминуче супроводжується логічними помилками. VS Code надає вбудований графічний відладчик (Debugger), який став незамінним інструментом у роботі.

Процес налагодження системи:

Під час реалізації класу FlowManager виникали ситуації, коли вектори ознак формувалися некоректно. Використання відладчика дозволило:

- Встановлювати точки зупину (breakpoints): Зупиняти виконання програми в момент обробки конкретного пакета.
- Інспектувати змінні: Переглядати вміст об`єктів у пам`яті без необхідності використання численних команд print().
- Покрокове виконання: Відстежувати логіку виконання програми рядок за рядком, що дозволило виявити та виправити помилку з неправильною розмірністю масивів NumPy перед подачею їх у нейронну мережу.

Особливістю розробки систем мережевої безпеки є необхідність роботи з привілеями адміністратора.

VS Code має вбудований термінал, який підтримує різні оболонки (PowerShell, Command Prompt, Bash).

- Керування правами: VS Code дозволяє запускати сесію терміналу з правами адміністратора. Це дало змогу запускати скрипт main.py командою python main.py безпосередньо з середовища розробки, отримуючи доступ до “живого” трафіку.
- Багатозадачність: Можливість відкривати декілька терміналів одночасно дозволила в одному вікні запустити main.py (захист), а в іншому - генерувати тестовий трафік або встановлювати додаткові бібліотеки через pip, не зупиняючи основний процес.

Для обґрунтування вибору VS Code було проведено порівняння з іншими популярними середовищами.

Таблиця 2.1 – Порівняння середовищ програмування

Характеристика	**Visual Studio Code**	**PyCharm (JetBrains)**	**Jupyter Lab (Browser)**
Тип	Редактор коду + Розширення	Повноцінна IDE	Веб-середовище
Споживання ресурсів	Середнє (Electron)	Високе (Java JVM)	Низьке
Робота з .ipynb	Відмінна (Вбудована)	Обмежена (у Community версії)	Рідна (Найкраща)
Робота з .py	Відмінна	Відмінна	Незручна
Швидкість запуску	Висока	Низька	Висока
Ціна	Безкоштовно (Open Source)	Є платна Pro версія	Безкоштовно

Хоча Jupyter Lab ідеально підходить для експериментів, він не пристосований для написання модульних програм (.py файлів). PyCharm є потужним інструментом, але його безкоштовна версія має обмежену підтримку наукових інструментів (Jupyter), а сама IDE споживає значно більше ресурсів, що критично при навчанні нейромереж на локальній машині. VS Code забезпечив “золоту середину”, поєднавши переваги обох світів.

Розробка програмного забезпечення вимагає надійного контролю змін. VS Code має вбудовану підтримку системи контролю версій Git. У ході виконання роботи це дозволило:

- Створювати коміти робочого коду після кожного успішного етапу (наприклад, “Виправлено архітектуру Autoencoder”).
- Експериментувати з різними підходами (наприклад, перемикання між MSE та MAE функціями втрат) у різних гілках, не боячись зламати основну робочу версію програми.
- Візуально відстежувати зміни в коді, що допомагало аналізувати, які саме модифікації призвели до покращення або погіршення метрик виявлення атак.

Використання Visual Studio Code як основного середовища розробки стало важливим фактором успішної реалізації системи виявлення вторгнень. Його гнучкість дозволила створити єдиний робочий простір для всіх задач: від аналізу даних у Jupyter Notebook до написання та налагодження складного мережевого сніфера.

Інтеграція з Python, підтримка віртуальних середовищ та зручні інструменти налагодження дозволили зосередитися на вирішенні наукової задачі - підвищенні точності виявлення аномалій, мінімізувавши час, витрачений на боротьбу з технічними проблемами налаштування середовища [24].

2.2 Попередня обробка та векторизації даних

Успіх роботи будь-якої системи машинного навчання на 80% залежить від якості вхідних даних. Оскільки набір даних CIC-IDS 2017 містить понад 2.8 мільйона записів із різнорідними, а іноді й пошкодженими даними, етап

попередньої обробки став одним із найскладніших та найважливіших у роботі. Для вирішення цих задач було використано зв`язку бібліотек NumPy та Pandas.

NumPy (Numerical Python) - це базова бібліотека для наукових обчислень у Python, яка надає підтримку багатовимірних масивів та матриць, а також велику колекцію високорівневих математичних функцій для операцій над цими масивами.

У той час як стандартні списки Python є гнучкими, вони повільні для математичних операцій через динамічну типізацію та розрізнене розташування об`єктів у пам`яті. NumPy вирішує цю проблему, вводячи об`єкт ndarray (n-dimensional array).

Продуктивність та Векторизація: Масиви NumPy зберігаються у безперервних блоках пам`яті, що дозволяє процесору ефективно використовувати кеш. Але головною перевагою є векторизація - здатність виконувати математичні операції над цілими масивами без використання повільних циклів for. *У* проєкті це було критично важливо при розрахунку помилки реконструкції (MAE) для Autoencoder.

Замість того, щоб перебирати кожен пакет у циклі, використовувалася векторизована формула:

$$MAE_i = \frac{1}{n}\sum_{j=1}^{n}\left|x_{ij} - \hat{x}1_{ij}\right|$$

Де MAE - середня абсолютна помилка для і-го рядка, n - кількість ознак, x_{ij} *- оригінальне значення j-ї ознаки для і-го пакету,* $\hat{x}1_{ij}$ *- відновлене значення j-ї ознаки, |...| - модуль різниці.*

NumPy дозволяє чітко контролювати типи даних, що важливо для зменшення споживання оперативної пам`яті при обробці великих датасетів.

У розробленій системі NumPy використовується для:

- Трансформації даних: Перетворення табличних даних Pandas у матриці, зрозумілі для TensorFlow та Scikit-learn.
- Статистичного аналізу: Розрахунок порогу аномальності за допомогою функції np.quantile(). Це дозволило автоматично визначити межу, яка відсікає 95% нормального трафіку, адаптуючи систему під конкретну мережу.
- Обробки невизначеностей: Використання констант np.inf та np.nan для ідентифікації та заміни некоректних значень у вхідному трафіку.[25]

Бібліотека Pandas використовується для маніпуляцій зі структурованими даними. Вона вводить поняття DataFrame - двовимірної табличної структури даних з мітками. Це бібліотека, побудована поверх NumPy, яка надає структури даних високого рівня для зручної роботи з табличними даними.

Датасет CIC-IDS 2017 розповсюджується у форматі CSV і складається з 8 окремих файлів, що містять трафік за різні дні тижня. Загальний обсяг даних перевищує кілька гігабайт. Pandas стала ідеальним інструментом для менеджменту цих даних завдяки наступним можливостям:

- Ефективне завантаження (I/O Tools): Функція pd.read_csv() є однією з найшвидших реалізацій парсерів CSV у світі Data Science. Вона дозволила автоматично розпізнати заголовки стовпців (78 ознак) та типи даних.
- Об`єднання даних (Merge and Concat): Для створення повноцінного тестового набору було використано функцію pd.concat(). Це дозволило програмно зібрати дані з 7 різних файлів (Вівторок-П`ятниця) в один гігантський DataFrame (df_test) для всебічної перевірки моделі, тоді як файл “Monday” був відокремлений для навчання (df_train).

```
df_train = pd.read_csv(os.path.join(path_to_data, "Monday-WorkingHours.pcap_ISCX.csv"))

all_files = glob.glob(os.path.join(path_to_data, "*.csv"))
test_files = [f for f in all_files if "Monday-WorkingHours" not in f]

print("Завантаження тестових даних (інші 7 файлів)")
list_of_dfs = []
for filename in test_files:
    print(f"Завантажується файл: {filename}")
    df_temp = pd.read_csv(filename)
    list_of_dfs.append(df_temp)

df_test = pd.concat(list_of_dfs, ignore_index=True)
```

Рисунок 2.1 - Об`єднання даних

Одним із найскладніших викликів у роботі стала наявність “брудних” даних у вихідному датасеті. Стандартні методи очищення не спрацьовували, що призводило до помилок масштабування (значення e+14). Засобами Pandas було реалізовано багаторівневий алгоритм очищення:

1. Нормалізація назв стовпців: Використання методу .str.strip() для видалення зайвих пробілів у назвах колонок (наприклад, ` Label` -> `Label`), що усунуло помилки доступу по ключу (KeyError).

2. Примусове перетворення типів: У деяких числових колонках (наприклад, Flow Bytes/s) зустрічалися текстові значення “Infinity”. Метод pd.to_numeric(..., errors=`coerce`) дозволив примусово перетворити ці стовпці на числа, автоматично замінюючи текст, що не розпізнається, на маркер NaN (Not a Number).

3. Обробка нескінченностей та пропусків: Було використано ланцюжок методів для заміни системних нескінченностей на NaN, з подальшим видаленням усіх пошкоджених рядків методом dropna().

```
X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_train.dropna(inplace=True)
```

Рисунок 2.2 – Обробка рядків

Цей підхід забезпечив математичну стабільність даних перед їх подачею на вхід нейронної мережі, що стало ключовим фактором досягнення високої точності виявлення атак.

Ще однією важливою функцією Pandas, використаною в роботі, є автоматичне вирівнювання даних за індексом. При видаленні “брудних” рядків з матриці ознак (X), відповідні мітки класів (y) також мають бути видалені. Pandas дозволяє зробити це елегантно через механізм індексації: y = y.loc[X.index]. Це гарантує, що мітки завжди відповідають своїм даним, запобігаючи зміщенню вибірки, яке могло б катастрофічно вплинути на навчання моделі.

У розробленій системі бібліотеки Pandas та NumPy виступають сполучною ланкою між “сирими” даними та алгоритмами штучного інтелекту.

1. Етап навчання (program.ipynb):

- Дані завантажуються в Pandas DataFrame.
- Проходять очищення та розділення на X та y.
- Конвертуються в NumPy масиви для обробки скейлером (MinMaxScaler).

2. Етап експлуатації:

- Хоча в реальному часі ми не використовуємо важкий DataFrame для кожного пакету (це було б повільно), ми використовуємо логіку NumPy для формування векторів ознак.

Використання Pandas та NumPy дозволило вирішити проблему “великих даних” у рамках магістерської роботи. Завдяки векторизації NumPy вдалося досягти високої швидкості математичних операцій, необхідних для роботи нейромережі. Гнучкість Pandas дозволила розробити надійний алгоритм очищення даних, який усунув критичні помилки у вхідному датасеті, що, у свою чергу, уможливило успішне навчання моделей Autoencoder та Isolation Forest.[26]

2.3 Побудова та навчання моделей машинного навчання

Бібліотека Scikit-learn (sklearn) є однією з найпотужніших та найпоширеніших бібліотек для машинного навчання на мові Python. Вона побудована на базі NumPy, SciPy та Matplotlib і надає прості та ефективні інструменти для інтелектуального аналізу даних. У рамках розробки системи ML-NADS ця бібліотека виконувала три критично важливі функції: нормалізацію даних, реалізацію ансамблевого алгоритму виявлення аномалій та оцінку ефективності всіх моделей.

Однією з головних проблем при роботі з мережевим трафіком є різний масштаб ознак. Наприклад, у датасеті CIC-IDS 2017 ознака Destination Port може набувати значень від 0 до 65535, Flow Duration - вимірюватися в мільйонах мікросекунд, а Fwd Packet Length - у сотнях байт. Якщо подати такі “сирі” дані на вхід нейронної мережі або алгоритмів, що базуються на відстанях, ознаки з великими числовими значеннями будуть домінувати над ознаками з малими значеннями, що призведе до некоректного навчання та повільної збіжності градієнтного спуску .

Для вирішення цієї проблеми засобами Scikit-learn було реалізовано модуль sklearn.preprocessing. Після серії експериментів з StandardScaler та RobustScaler

(які показали нестабільність на даних з екстремальними викидами), було обрано алгоритм MinMax Scaling.

Алгоритм MinMaxScaler:

Цей метод трансформує кожну ознаку таким чином, щоб вона знаходилася у заданому діапазоні, зазвичай [0, 1]. Математично це описується формулою:

$$X\_std = (X - X\_min)/(X\_\max - X\_min)$$

$$X_{scaled} = X_{std} \cdot (\max - min)$$

Переваги вибору для системи виявлення вторгнень:

1. Сумісність з нейромережею: Вихідний діапазон [0, 1] ідеально узгоджується з функцією активації sigmoid, яка використовується у вихідному шарі розробленого Autoencoder. Це дозволяє моделі коректно “реконструювати” вхідні дані.

2. Збереження розподілу: На відміну від стандартизації, MinMax-нормалізація не спотворює розподіл даних, а лише змінює їхній масштаб, що важливо для збереження патернів атак.

```
scaler = MinMaxScaler()

scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Дані масштабовано")
```

Рисунок 2.3 – Масштабування даних

Для порівняльного аналізу та створення гібридної системи захисту, окрім нейронної мережі, було використано класичний алгоритм машинного навчання без учителя - Isolation Forest, реалізований у класі sklearn.ensemble.IsolationForest.

На відміну від традиційних методів, які намагаються побудувати профіль “нормальної” поведінки (як One-Class SVM), Isolation Forest явно ізолює аномалії. Алгоритм будує ансамбль бінарних дерев рішень. Для побудови кожного дерева випадковим чином вибирається ознака і випадкове значення розгалуження (split value) між мінімумом і максимумом цієї ознаки.

Логіка базується на тому, що аномалії (мережеві атаки) є “рідкісними і відмінними” (few and different):

1. Оскільки аномальні точки даних відрізняються від нормальних, вони швидше потрапляють у “листя” дерев (потребують меншої кількості розгалужень для ізоляції).

2. Нормальні точки, які утворюють щільні кластери, потребують значно більшої кількості розгалужень.

3. “Оцінка аномальності” розраховується як середня довжина шляху від кореня до листа по всіх деревах ансамблю. Чим коротший шлях, тим вища ймовірність атаки.

Налаштування параметрів у Scikit-learn:

У роботі було використано наступні параметри ініціалізації моделі:

- n_estimators=100: Кількість дерев у лісі. Експериментально встановлено, що 100 дерев забезпечують баланс між точністю та швидкістю роботи.

- contamination: Параметр, що визначає очікувану частку аномалій у навчальному наборі. У фінальній версії системи цей параметр було встановлено

на рівні 0.01 (1%), що відображає припущення про чистоту навчальної вибірки (Monday dataset) з можливістю наявності незначного статистичного шуму.

- n_jobs=-1: Дозволяє використовувати всі ядра процесора для паралельної побудови дерев, що критично важливо при навчанні на великих датасетах [27].

Для об`єктивної оцінки ефективності розробленої системи NIDS було використано модуль sklearn.metrics. Оскільки задача виявлення вторгнень є задачею бінарної класифікації на незбалансованих даних, використання простої метрики “Точність” (Accuracy) є недостатнім. Scikit-learn надає інструменти для глибокого аналізу помилок.

1. Матриця похибок (Confusion Matrix):

Функція confusion_matrix дозволила візуалізувати чотири ключові показники:

- True Negatives (TN): Нормальний трафік, коректно класифікований як норма.
- False Positives (FP): Нормальний трафік, помилково класифікований як атака (“Хибна тривога”).
- False Negatives (FN): Атака, пропущена системою (найбільш небезпечний тип помилки).
- True Positives (TP): Атака, успішно виявлена системою.

2. Звіт класифікації (Classification Report):

Функція classification_report автоматично розраховує метрики, які є стандартом у кібербезпеці:

- Precision (Точність): TP / (TP + FP). Показує, наскільки можна довіряти сигналу тривоги.

- Recall (Повнота): TP / (TP + FN). Показує, яку частку всіх реальних атак система змогла виявити.
- F1-Score: Гармонійне середнє між Precision та Recall. Ця метрика стала основною для порівняння ефективності Isolation Forest та Autoencoder у даній роботі.

Scikit-learn також забезпечив інфраструктуру для управління життєвим циклом даних.

Функція model_selection.train_test_split використовувалася на початкових етапах для розділення даних на навчальну та валідаційну вибірки, що дозволило уникнути ефекту перенавчання.

Важливим аспектом впровадження системи є збереження параметрів попередньої обробки. Об`єкт MinMaxScaler, навчений на “чистому” трафіку, зберігає в собі параметри min та max для кожної з 78 ознак. За допомогою бібліотеки Joblib (яка є рекомендованим способом серіалізації моделей Scikit-learn) цей об`єкт було збережено у файл scaler.pkl. Це дозволило модулю main.py використовувати ідентичне масштабування для нових пакетів у реальному часі, що є обов`язковою умовою коректної роботи системи.

Бібліотека Scikit-learn відіграла роль фундаментального інструменту для побудови класичних моделей та підготовки даних. Її модульність та сумісність з іншими бібліотеками (Pandas, TensorFlow) дозволили створити єдиний конвеєр (pipeline) обробки даних.[28]

Використання MinMaxScaler забезпечило необхідну підготовку даних для нейронної мережі, а реалізація Isolation Forest надала важливу точку відліку (baseline) для оцінки ефективності більш складних методів глибокого навчання. Інструменти метрик дозволили провести науково обґрунтовану валідацію результатів.

У той час як класичні алгоритми (такі як Isolation Forest) демонструють високу швидкість, задачі виявлення складних мережевих атак вимагають здатності моделювати нелінійні залежності у багатовимірному просторі ознак. Для реалізації такого підходу в роботі було використано методи глибокого навчання (Deep Learning).

В якості технологічної основи обрано фреймворк TensorFlow (розробка Google) та його високорівневий API Keras. Ця комбінація є індустріальним стандартом для побудови нейронних мереж завдяки своїй гнучкості, масштабованості та продуктивності.

TensorFlow - це відкрита програмна бібліотека для чисельного обчислення з використанням графів потоків даних (data flow graphs). Вузли графа представляють математичні операції, а ребра - багатовимірні масиви даних (тензори), які передаються між ними.

Keras - це інтерфейс прикладного програмування (API), який працює поверх TensorFlow. Він дозволяє конструювати нейронні мережі як конструктор, оперуючи поняттями Layer, Model, Optimizer, приховуючи складні математичні перетворення тензорів “під капотом”.

Обґрунтування вибору:

- Sequential API: Для реалізації Autoencoder було використано послідовну модель (keras.models.Sequential або функціональний API), що дозволяє лінійно нанизувати шари енкодера та декодера.
- Автоматичне диференціювання: TensorFlow автоматично обчислює градієнти для зворотного поширення помилки (Backpropagation), що звільняє дослідника від необхідності вручну реалізовувати складні похідні функцій втрат.

- Eager Execution: Режим негайного виконання дозволяє оцінювати операції відразу, без побудови графів, що спростило налагодження моделі та аналіз проміжних значень тензорів під час розробки.

Центральним елементом інтелектуальної системи захисту є Автокодувальник. Це специфічний тип нейронної мережі, який навчається копіювати вхідні дані на вихід, проходячи через “вузьке горло” (bottleneck) - шар зі зменшеною розмірністю.

У модулі навчання (program.ipynb) розроблено та реалізовано наступну топологію мережі:

1. Вхідний шар (Input Layer):

Приймає вектор із 78 ознак, що відповідає структурі даних після попередньої обробки (очищення та нормалізація датасету CIC-IDS 2017).

2. Енкодер (Encoder):

Серія повнозв'язних шарів (Dense), що поступово зменшують розмірність даних. Мета енкодера - вилучити найбільш значущі латентні ознаки нормального трафіку, відкинувши шум.

- Шар 1: 64 нейрони, функція активації ReLU.
- Шар 2: 32 нейрони, функція активації ReLU.
- Шар 3 (Bottleneck): 16 нейронів, функція активації ReLU.

Вибір ReLU (Rectified Linear Unit*):* Використання функції $f(x) = \max(0, x)$ дозволяє уникнути проблеми зникаючого градієнта та прискорити збіжність моделі порівняно з гіперболічним тангенсом [29].

3. Декодер (Decoder):

Дзеркальна структура, що намагається відновити початкові дані зі стисненого представлення (з 16 ознак назад до 78).

- Шар 4: 32 нейрони, активація ReLU.
- Шар 5: 64 нейрони, активація ReLU.

4. Вихідний шар (Output Layer):

Містить 78 нейронів. На цьому шарі використано функцію активації Sigmoid.

Оскільки вхідні дані були нормалізовані за допомогою MinMaxScaler у діапазон [0, 1], вихід мережі також повинен бути обмежений цим діапазоном. Використання лінійної активації або ReLU на виході призводило б до некоректних значень реконструкції та неможливості навчання моделі.

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 78)	0
dense_6 (Dense)	(None, 64)	5,056
dense_7 (Dense)	(None, 32)	2,080
dense_8 (Dense)	(None, 16)	528
dense_9 (Dense)	(None, 32)	544
dense_10 (Dense)	(None, 64)	2,112
dense_11 (Dense)	(None, 78)	5,070

Рисунок 2.4 – Архітектура Autoencoder

Вибір функції, яка оцінює якість реконструкції, став одним із найскладніших етапів дослідження.

Експеримент 1: Mean Squared Error (MSE)

Спочатку використовувалася середньоквадратична помилка:

$$MSE = \frac{1}{n}\sum(Y_i - \hat{Y}1_i)^2$$

Однак, експерименти показали, що наявність навіть незначних статистичних викидів (outliers) у "нормальному" трафіку (наприклад, рідкісні пакети великого розміру) призводила до того, що MSE "вибухала". Квадратична функція надає надмірну вагу великим помилкам, змушуючи модель фокусуватися на відтворенні викидів, а не загальної структури трафіку.

Експеримент 2: Mean Absolute Error (MAE)

Для вирішення цієї проблеми було здійснено перехід на середню абсолютну помилку:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}1_i|$$

Результат: MAE виявилася стійкою (robust) до викидів. Вона лінійно штрафує помилки, що дозволило стабілізувати процес навчання та досягти значення функції втрат < 0.01 на валідаційній вибірці. Саме MAE використовується в фінальній системі (main.py) як міра аномальності: якщо MAE пакету перевищує поріг, він вважається атакою.

Процес навчання моделі (training loop) був реалізований за допомогою методу .fit() бібліотеки Keras у середовищі Jupyter Notebook (program.ipynb).

1. Оптимізатор Adam (Adaptive Moment Estimation):

Використано алгоритм Adam, який комбінує ідеї RMSProp та Momentum. Він автоматично адаптує швидкість навчання (learning rate) для кожного параметра окремо. Це дозволило моделі швидко знайти глобальний мінімум функції втрат без необхідності ручного налаштування кроку градієнтного спуску.

2. Параметри навчання:

- Епохи (Epochs): 20. Експериментально встановлено, що 20 повних проходів по датасету достатньо для стабілізації функції втрат без ризику перенавчання (overfitting).

- Розмір пакету (Batch Size): 32. Використання міні-пакетів (mini-batch gradient descent) забезпечило баланс між швидкістю обчислень та точністю оновлення ваг.

- Shuffle: Перемішування даних перед кожною епохою запобігає “запам'ятовуванню” порядку прикладів моделлю.

```
Навчання Autoencoder
Epoch 1/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 76s 4ms/step - loss: 0.0134
Epoch 2/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 88s 5ms/step - loss: 0.0062
Epoch 3/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 87s 5ms/step - loss: 0.0051
Epoch 4/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 138s 5ms/step - loss: 0.0047
Epoch 5/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 144s 5ms/step - loss: 0.0046
Epoch 6/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 139s 5ms/step - loss: 0.0045
Epoch 7/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 56s 3ms/step - loss: 0.0045
Epoch 8/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 87s 4ms/step - loss: 0.0044
Epoch 9/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 70s 4ms/step - loss: 0.0044
Epoch 10/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 69s 4ms/step - loss: 0.0044
Epoch 11/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 62s 4ms/step - loss: 0.0043
Epoch 12/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 62s 4ms/step - loss: 0.0043
...
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 51s 3ms/step - loss: 0.0042
Epoch 20/20
16547/16547 ━━━━━━━━━━━━━━━━━━━━ 55s 3ms/step - loss: 0.0042
```

Рисунок 2.5 – Навчання моделі Autoencoder

Після завершення навчання модель необхідно було експортувати для використання у модулі детектування реального часу (main.py). TensorFlow надає для цього формат HDF5 (.h5).

Метод autoencoder.save('autoencoder_full.h5') зберігає:

1. Архітектуру моделі (кількість шарів, нейронів).
2. Ваги всіх нейронів (отримані в результаті навчання).

3. Конфігурацію оптимізатора (дозволяє продовжити навчання з того ж місця).

Це забезпечило можливість “гарячого” завантаження моделі у скрипт main.py за допомогою функції load_model() без необхідності витрачати час на повторне навчання при кожному запуску системи захисту. Цей підхід дозволяє чітко розділити фазу навчання (offline) та фазу експлуатації (online).

Використання TensorFlow та Keras дозволило реалізувати складну архітектуру глибокого автокодувальника, здатного вивчати приховані патерни нормальної поведінки мережі.[29,30]

Ключовим досягненням на цьому етапі стала оптимізація архітектури:

- Використання MinMax масштабування у поєднанні з Sigmoid активацією для узгодження діапазонів даних.
- Заміна функції втрат MSE на MAE для підвищення стійкості до статистичних викидів у навчальних даних.

Це дозволило створити модель, яка демонструє високу чутливість до аномалій (NIDS), що підтверджено експериментально в наступних розділах.

2.4 Візуалізація результатів та серіалізації моделей

Завершальним і критично важливим етапом розробки програмного комплексу є забезпечення його автономності, відтворюваності результатів та можливості інтерпретації отриманих даних. Цей етап охоплює реалізацію механізмів збереження навчених параметрів системи для їх подальшого використання в режимі реального часу, а також застосування засобів графічної візуалізації для наукової валідації ефективності моделей.

Для забезпечення цілісності робочого процесу та можливості роботи системи в режимі реального часу було реалізовано механізм персистентності, оскільки процес тренування нейронної мережі та статистичних моделей на масиві даних, що перевищує 2.8 мільйона записів, є вкрай ресурсомістким і вимагає значних часових витрат.

Щоб уникнути необхідності повторного навчання при кожному запуску скрипта детектування, було використано спеціалізовані бібліотеки серіалізації. Для збереження об'єктів екосистеми Scikit-learn, зокрема моделі Isolation Forest та об'єктів попередньої обробки, було обрано бібліотеку Joblib, яка є більш ефективною альтернативою стандартному модулю pickle при роботі з об'єктами, що містять великі масиви даних NumPy. За допомогою цієї бібліотеки навчена модель Isolation Forest експортується у бінарний файл, зберігаючи структуру дерев рішень та порогові значення для виявлення аномалій.

Окремим і, можливо, найважливішим аспектом є збереження об'єкта нормалізації, оскільки екземпляр класу MinMaxScaler під час навчання фіксує мінімальні та максимальні значення для кожної з 78 ознак тренувальної вибірки, і для коректної класифікації нових пакетів у реальному часі необхідно використовувати той самий екземпляр скейлера з ідентичними параметрами, що гарантує математичну консистентність даних між етапом навчання та етапом експлуатації.

Для збереження моделі глибокого навчання Autoencoder було використано нативний формат бібліотеки Keras - HDF5, який дозволяє в одному файлі зберегти повну архітектуру мережі, включаючи кількість шарів, нейронів та функції активації, а також матриці вагових коефіцієнтів та зміщень, отримані в результаті оптимізації.

Паралельно із забезпеченням автономності системи, для наукової валідації отриманих результатів та глибокої оцінки якості роботи алгоритмів було задіяно бібліотеки візуалізації даних Matplotlib та Seaborn, які дозволили трансформувати сухі числові метрики у наочні графічні представлення. Зокрема, за допомогою Matplotlib було побудовано графіки динаміки функції втрат, що візуалізують зміну помилки реконструкції на тренувальній та валідаційній вибірках протягом епох навчання, дозволяючи підтвердити збіжність моделі, діагностувати відсутність ефектів перенавчання або недонавчання та обґрунтувати вибір кількості епох, достатньої для стабілізації ваг нейромережі.

Для детального аналізу якості класифікації було використано можливості бібліотеки Seaborn для побудови теплових карт, що дозволило візуалізувати матрицю похибок і наочно продемонструвати співвідношення істинно-позитивних, істинно-негативних, хибно-позитивних та хибно-негативних результатів, надаючи можливість миттєво оцінити слабкі місця моделі. Крім того, для обґрунтування вибору порогу спрацювання для автокодувальника було використано гістограми розподілу помилки реконструкції, які графічно підтвердили чітке розмежування між класами, де нормальний трафік групується в зоні малих помилок, а атаки утворюють характерний «хвіст» розподілу з високими значеннями, що стало остаточним доказом здатності моделі розрізняти семантику мережевих пакетів.[30,31]

Висновки до розділу:

У другому розділі магістерської роботи здійснено наукове обґрунтування та теоретичне проєктування інтелектуальної системи виявлення мережевих

аномалій ML-NADS. За результатами проведених досліджень формалізовано постановку задачі виявлення вторгнень як задачу навчання без учителя, що дозволяє ефективно ідентифікувати невідомі раніше загрози та атаки нульового дня шляхом аналізу статистичних відхилень від еталонної моделі нормальної поведінки мережі.

В якості технологічного підґрунтя обґрунтовано вибір мови програмування Python та відповідного стеку бібліотек, зокрема Pandas і NumPy для високоефективної обробки та векторизації великих масивів даних, Scikit-learn для реалізації методу Isolation Forest та попередньої обробки, а також фреймворку TensorFlow/Keras для побудови моделей глибокого навчання.

Центральним елементом розробленої архітектури стала спроєктована модель глибокого автокодувальника з оптимізованою топологією, функціями активації ReLU та Sigmoid, а також стійкою до викидів функцією втрат Mean Absolute Error.

Крім того, сформовано математичну модель попередньої обробки даних, що базується на методі нормалізації MinMax Scaling та стратегії One-Class Learning, яка передбачає навчання системи виключно на легітимному трафіку для формування профілю норми.

Розроблена структурна схема системи описує взаємодію двох ключових компонентів: модуля навчання, який відповідає за генерацію та збереження моделей, та модуля детектування, що здійснює сніфінг трафіку та прийняття рішень на основі розрахованого адаптивного порогу, створюючи необхідну теоретико-методологічну базу для програмної реалізації та експериментального дослідження.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ СИСТЕМИ ВИЯВЛЕННЯ МЕРЕЖЕВИХ АНОМАЛІЙ

3.1 Програмна реалізація етапу підготовки даних та навчання моделей

Для реалізації та експериментального дослідження системи виявлення вторгнень було використано апаратну конфігурацію, що складається з 16 ГБ оперативної пам'яті та відеокартуNVIDIA GeForce RTX 3050 з 6 ГБ відеопам'яті. Такий обсяг оперативної пам'яті є критично важливим для етапу попередньої обробки даних, оскільки використаний набір даних CIC-IDS 2017 є надзвичайно об'ємним і містить понад 2,8 мільйона записів, кожен з яких описується 78 статистичними ознаками. Завантаження повного обсягу цих даних у структуру DataFrame бібліотеки Pandas, їх очищення та виконання операцій нормалізації вимагає одночасного утримання в пам'яті значних масивів інформації, і саме наявність 16 ГБ дозволяє уникнути використання повільного файлу підкачки та забезпечує стабільність роботи середовища розробки під час маніпуляцій з даними.

У свою чергу, наявність дискретної відеокарти NVIDIA GeForce RTX 3050 стала визначальним фактором для ефективного навчання моделі глибокого автокодувальника. Завдяки підтримці архітектури паралельних обчислень CUDA, бібліотека TensorFlow автоматично переносить основне обчислювальне навантаження, пов'язане з тисячами матричних операцій та зворотним поширенням помилки, з центрального процесора на графічні ядра відеокарти. Відеопам'ять обсягом 6 ГБ при цьому використовується для зберігання ваг нейронної мережі та пакетів даних безпосередньо під час ітерацій навчання.

Таке апаратне прискорення дозволило скоротити час проходження однієї епохи навчання з годин до лічених секунд, що дало можливість провести широку серію експериментів з налаштування гіперпараметрів моделі та досягти оптимальних показників точності виявлення атак.

Розробка програмного комплексу розпочинається з ініціалізації середовища та підключення необхідних бібліотек, серед яких ключову роль відіграють Pandas для маніпуляції табличними даними, NumPy для лінійної алгебри, а також системні модулі os та glob для взаємодії з файловою системою. У першій комірці коду реалізовано стратегію розділення даних, яка є фундаментальною для обраного методу виявлення аномалій. Оскільки архітектура автокодувальника передбачає навчання виключно на легітимному трафіку (One-Class Learning), для формування тренувальної вибірки df_train було цілеспрямовано обрано файл Monday-WorkingHours.pcap_ISCX.csv. Цей вибір обґрунтований тим, що згідно з документацією датасету CIC-IDS 2017, трафік першого дня експерименту не містить жодних кібератак, що дозволяє моделі сформувати еталонний профіль нормальної поведінки мережі без шумів та аномальних вкраплень.

Паралельно з цим відбувається формування тестової вибірки df_test, яка має відображати реальні умови функціонування системи, де зустрічаються як нормальні пакети, так і різноманітні загрози. За допомогою бібліотеки glob здійснюється автоматичний пошук усіх наявних CSV-файлів у директорії з даними, після чого програмний алгоритм відфільтровує файл понеділка, щоб уникнути дублювання даних у навчальному та тестовому наборах і забезпечити чистоту експерименту. Решта файлів, що містять записи трафіку з вівторка по п`ятницю та включають широкий спектр атак (DDoS, PortScan, Web Attacks, Infiltration), послідовно зчитуються та агрегуються у тимчасовий список. Завершальною операцією цього етапу є конкатенація всіх завантажених

фрагментів в єдиний масив даних за допомогою функції pd.concat з параметром ігнорування індексів, що забезпечує цілісність структури отриманого датафрейму. Такий підхід дозволив отримати два чітко розмежовані набори даних: “чисту” тренувальну вибірку обсягом понад 500 тисяч записів для навчання нейронної мережі та репрезентативну тестову вибірку обсягом понад 2.3 мільйона записів для об`єктивної перевірки її ефективності на невідомих раніше даних.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import glob
import os

path_to_data = 'Data'

print("Завантаження тренувальних даних (Monday)")
df_train = pd.read_csv(os.path.join(path_to_data, "Monday-WorkingHours.pcap_ISCX.csv"))

all_files = glob.glob(os.path.join(path_to_data, "*.csv"))
test_files = [f for f in all_files if "Monday-WorkingHours" not in f]

print("Завантаження тестових даних (інші 7 файлів)")
list_of_dfs = []
for filename in test_files:
    print(f"Завантажується файл: {filename}")
    df_temp = pd.read_csv(filename)
    list_of_dfs.append(df_temp)

df_test = pd.concat(list_of_dfs, ignore_index=True)

print("\nДані успішно розділені на df_train та df_test.")
print(f"Розмір тренувальних даних (df_train): {df_train.shape}")
print(f"Розмір тестових даних (df_test): {df_test.shape}")
```

Рисунок 3.1 - Ініціалізація середовища

Після успішного завантаження та розділення масивів даних на тренувальну та тестову вибірки, наступним критичним етапом стала їхня глибока попередня

обробка. “Сирі” дані мережевого трафіку часто містять шуми, некоректні значення або формати, які унеможливлюють пряме застосування математичних алгоритмів. У другій комірці коду реалізовано комплексний підхід до очищення та трансформації даних, який розпочинається з стандартизації назв атрибутів. Використання методу .str.strip() для об`єктів стовпців дозволило автоматично видалити зайві пробіли на початку та в кінці назв змінних, що є поширеною проблемою в CSV-файлах і може призводити до помилок доступу за ключем у подальшому. Далі було виконано розділення наборів даних на матрицю ознак та вектор цільових міток Для тренувальної вибірки, яка складається виключно з нормального трафіку, вектор міток формально зберігається для контролю, хоча в процесі навчання Autoencoder він не використовується напряму.

Ключовим етапом підготовки стала процедура очищення числових даних. Оскільки датасет CIC-IDS 2017 містить змішані типи даних та специфічні позначення помилок, було застосовано функцію pd.to_numeric з параметром errors=`coerce`. Це дозволило примусово перетворити всі значення ознак у числовий формат, автоматично замінюючи будь-які текстові вкраплення або помилки на стандартний маркер відсутності даних (NaN). Окрему увагу було приділено обробці нескінченних значень (Infinity), які часто виникають при розрахунку статистичних показників (наприклад, при діленні на нульову тривалість потоку). Такі значення були програмно замінені на NaN, після чого всі рядки, що містили хоча б одне пропущене або некоректне значення, були видалені з вибірки методом dropna. Важливо зазначити, що після видалення “брудних” рядків з матриці ознак X, було виконано синхронізацію вектора міток у за індексом, щоб зберегти цілісність відповідності між даними та їх класифікацією. Аналогічна процедура очищення була дзеркально застосована і до тестового набору даних, що гарантує однакову структуру вхідних векторів на етапах навчання та експлуатації.

Завершальним і одним з найважливіших кроків підготовки даних стала їх нормалізація. Оскільки нейронні мережі, зокрема автокодувальники, чутливі до масштабу вхідних даних, а ознаки мережевого трафіку мають різні діапазони вимірювання (від одиниць до мільйонів), було обрано метод MinMax-нормалізації. За допомогою класу MinMaxScaler з бібліотеки Scikit-learn усі числові значення були приведені до діапазону [0, 1]. Методологічно важливим є те, що навчання скейлера (метод fit) проводилося виключно на “чистих” тренувальних даних (X_train). Це дозволило зафіксувати параметри мінімуму та максимуму, характерні саме для нормальної поведінки мережі. Тестові дані (X_test) згодом трансформувалися (метод transform) на основі параметрів, отриманих з тренувальної вибірки. Такий підхід запобігає явищу “витоку даних” і гарантує, що модель сприйматиме аномальні відхилення в тестових даних саме як аномалії, що виходять за межі звичного масштабу нормального трафіку. У результаті виконання цього блоку коду було отримано два масиви - X_train_scaled та X_test_scaled, які є повністю готовими для подачі на вхід алгоритмів машинного навчання.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

df_train.columns = df_train.columns.str.strip()
df_test.columns = df_test.columns.str.strip()

print("Очищення тренувальних даних")
y_train = df_train['Label']
X_train = df_train.drop('Label', axis=1)

X_train = X_train.apply(pd.to_numeric, errors='coerce')
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_train.dropna(inplace=True)
y_train = y_train.loc[X_train.index]

print("Очищення тестових даних")
y_test = df_test['Label']
X_test = df_test.drop('Label', axis=1)

X_test = X_test.apply(pd.to_numeric, errors='coerce')
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.dropna(inplace=True)
y_test = y_test.loc[X_test.index]

print("Дані очищено та розділено.")

print("Масштабування даних (MinMaxScaler)")
scaler = MinMaxScaler()

scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Рисунок 3.2 – Обробка даних

Наступним етапом програмної реалізації стала побудова базової моделі виявлення аномалій на основі алгоритму Isolation Forest. У третій комірці коду здійснюється імпорт відповідного класу з бібліотеки Scikit-learn та ініціалізація моделі зі специфічними гіперпараметрами, адаптованими під умови навчання на «чистих» даних. Ключовим налаштуванням є параметр contamination, який

визначає очікувану частку аномалій у навчальній вибірці. Оскільки для тренування використовується трафік “Monday”, який вважається легітимним, значення цього параметра було встановлено на рівні 0.01 (1%). Це рішення обумовлене необхідністю врахування можливого статистичного шуму або незначних відхилень у нормальному трафіку, запобігаючи перенавчанню моделі на рідкісних, але безпечних подіях.

Для забезпечення відтворюваності результатів експерименту параметр random_state було зафіксовано на значенні 42, що гарантує ідентичність побудови дерев при кожному запуску коду. З метою оптимізації часу обчислень використано параметр n_jobs=-1, який дозволяє алгоритму задіяти всі доступні ядра центрального процесора для паралельної побудови ансамблю дерев. Процес навчання ініціюється методом fit, на вхід якого подається нормалізований масив X_train_scaled. Під час виконання цього методу алгоритм будує набір випадкових дерев рішень, які фіксують структуру нормального простору ознак. Точки даних, що відповідають нормальній поведінці мережі, потребують більшої кількості розгалужень для ізоляції, тоді як потенційні аномалії будуть ізольовані на ранніх етапах побудови дерева. Успішне завершення навчання підтверджується відповідним повідомленням у консолі, після чого об`єкт моделі model_iso готовий до використання для класифікації нових даних.

```
from sklearn.ensemble import IsolationForest

model_iso = IsolationForest(contamination=0.01, n_jobs=-1, random_state=42)

print("Навчання Isolation Forest (Monday)")

model_iso.fit(X_train_scaled)

print("Модель Isolation Forest навчена")
```

Рисунок 3.3 – Навчання Isolation Forest

Після завершення етапу навчання, критично важливим кроком є валідація розробленої моделі на тестовому наборі даних, який містить реальні приклади кібератак. У четвертій комірці коду реалізовано процедуру тестування, що розпочинається з генерації прогнозів для масиву X_test_scaled за допомогою методу predict. Специфіка алгоритму Isolation Forest у бібліотеці Scikit-learn полягає у тому, що він повертає значення -1 для аномалій та 1 для нормальних даних. Оскільки у вихідному датасеті мітки класів мають інший формат (текстові значення “BENIGN” та назви атак), було реалізовано програмний блок нормалізації міток. За допомогою спискового включення (list comprehension) та лямбда-функцій вихідні дані моделі та істинні мітки тестової вибірки були приведені до єдиного бінарного формату, де 0 відповідає нормальному трафіку, а 1 - атаці. Ця уніфікація дозволила застосувати стандартні метрики оцінки якості бінарної класифікації.

Для кількісної оцінки роботи моделі було розраховано загальну точність (Accuracy) за допомогою функції accuracy_score, яка відображає відсоток правильних передбачень. Однак, враховуючи дисбаланс класів у мережевому трафіку, більш інформативним є детальний звіт про класифікацію (classification_report). Ця функція генерує таблицю з ключовими метриками: точністю (Precision), повнотою (Recall) та F1-мірою для кожного класу окремо. Це дозволяє детально проаналізувати здатність моделі не лише виявляти атаки, але й мінімізувати кількість хибних спрацьовувань. Для візуальної інтерпретації результатів було побудовано матрицю похибок (Confusion Matrix). Використання бібліотеки Seaborn (sns.heatmap) дозволило представити матрицю у вигляді теплової карти, що наочно демонструє розподіл істинно-позитивних, істинно-негативних, хибно-позитивних та хибно-негативних результатів. Такий візуальний аналіз є необхідним для розуміння того, які саме типи помилок переважають у роботі статистичного алгоритму ізоляції.

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

y_pred_iso = model_iso.predict(X_test_scaled)

y_pred_iso_normalized = [1 if x == -1 else 0 for x in y_pred_iso]

y_test_normalized = y_test.apply(lambda x: 1 if x != 'BENIGN' else 0)

accuracy = accuracy_score(y_test_normalized, y_pred_iso_normalized)
print(f"Загальна точність моделі: {accuracy * 100:.2f}%\n")

print("Звіт по класифікації:")
print(classification_report(y_test_normalized, y_pred_iso_normalized, target_names=['Норма (0)', 'Атака (1)']))

print("Матриця похибок:")
cm = confusion_matrix(y_test_normalized, y_pred_iso_normalized)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Норма', 'Атака'], yticklabels=['Норма', 'Атака'])
plt.xlabel('Прогноз')
plt.ylabel('Реальний')
plt.show()
```

Рисунок 3.4 - Оцінка ефективності моделі Isolation Forest

Ключовим елементом розробленої системи є модуль глибокого навчання, реалізація якого наведена у восьмій комірці програмного коду. Для створення нейронної мережі типу “Автокодувальник” (Autoencoder) було використано високорівневий API Keras, що входить до складу бібліотеки TensorFlow. Архітектура мережі спроєктована за симетричним принципом і складається з двох функціональних блоків: енкодера та декодера. На вхідний шар подається вектор ознак, розмірність якого визначається автоматично на основі масштабованого тренувального набору X_train_scaled (78 нейронів). Енкодер послідовно стискає вхідні дані через серію повнозв`язних шарів (Dense) зі зменшенням кількості нейронів за схемою 64 -> 32 -> 16. Шар з 16 нейронами виступає в ролі “вузького горла” (bottleneck) або латентного простору, змушуючи мережу відкидати шум і вивчати лише найбільш суттєві патерни нормального мережевого трафіку. Для всіх прихованих шарів в якості функції активації обрано ReLU (Rectified Linear Unit), що забезпечує нелінійність моделі та ефективну боротьбу з проблемою зникаючого градієнта.

Декодер виконує зворотну операцію, відновлюючи розмірність даних від латентного представлення до початкового розміру вхідного вектора (16 > 32 > 64 > 78). Критично важливим рішенням на цьому етапі є вибір функції активації для вихідного шару. Оскільки на етапі попередньої обробки всі вхідні дані були нормалізовані у діапазон [0, 1] за допомогою MinMaxScaler, для вихідного шару було застосовано сигмоїдальну функцію активації (activation=`sigmoid`). Це гарантує, що реконструйовані дані також будуть знаходитися в межах від 0 до 1, що є математично необхідною умовою для коректного обчислення помилки реконструкції. Після визначення топології мережі було створено об`єкт моделі Model, який об`єднує вхідний та вихідний потоки даних.

Завершальним етапом конструювання є компіляція моделі, під час якої визначаються параметри процесу навчання. В якості оптимізатора обрано алгоритм Adam, який є стандартом для задач глибокого навчання завдяки адаптивному керуванню швидкістю навчання. Для функції втрат (Loss Function) було обрано середню абсолютну помилку (mean_absolute_error або MAE). Цей вибір, на відміну від класичної середньоквадратичної помилки (MSE), обумовлений необхідністю підвищення стійкості моделі до статистичних викидів. Оскільки навіть у “чистому” трафіку можуть траплятися пікові значення, використання MAE запобігає надмірному впливу окремих аномальних значень на градієнти під час навчання, забезпечуючи більш стабільну збіжність моделі. Виклик методу summary() в кінці блоку коду дозволяє візуалізувати структуру побудованої мережі та переконатися у коректності кількості параметрів, що підлягають навчанню.

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

input_dim = X_train_scaled.shape[1]

input_layer = Input(shape=(input_dim, ))

encoder = Dense(64, activation='relu')(input_layer)
encoder = Dense(32, activation='relu')(encoder)
encoder = Dense(16, activation='relu')(encoder)

decoder = Dense(32, activation='relu')(encoder)
decoder = Dense(64, activation='relu')(decoder)

decoder = Dense(input_dim, activation='sigmoid')(decoder)

autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.compile(optimizer='adam', loss='mean_absolute_error')

autoencoder.compile(optimizer='adam', loss='mean_absolute_error')

autoencoder.summary()
```

Рисунок 3.5 - Проєктування архітектури Autoencoder

Після затвердження архітектури та компіляції моделі, у дев`ятій комірці реалізовано безпосередній процес навчання нейронної мережі. Цей етап є найбільш ресурсомістким, оскільки він передбачає ітеративну оптимізацію вагових коефіцієнтів моделі для мінімізації помилки реконструкції. Для запуску процесу використано метод .fit() бібліотеки Keras, який приймає на вхід навчальні дані та параметри тренування.

Фундаментальною особливістю навчання автокодувальника, яка відображена в коді, є те, що вхідні дані та цільові значення (labels) є ідентичними. У виклику функції fit(X_train_scaled, X_train_scaled, ...) перший аргумент - це вхідні вектори ознак, а другий - це ті самі вектори, які модель повинна навчитися відтворювати на виході. Це реалізує парадигму самонавчання (Self-Supervised

Learning), де система вчиться стискати та відновлювати інформацію про нормальний трафік без необхідності ручного маркування аномалій.

Для керування процесом навчання було задано наступні гіперпараметри:

- Епохи (epochs=20): Цей параметр визначає, що нейронна мережа “побачить” повний набір навчальних даних 20 разів. Аналіз динаміки функції втрат показав, що цієї кількості ітерацій достатньо для досягнення стабільного мінімуму помилки (Loss), уникаючи при цьому ефекту перенавчання.
- Розмір пакету (batch_size=32): Навчання відбувається не на всьому масиві даних одночасно, а невеликими пакетами по 32 зразки. Це забезпечує частіше оновлення ваг градієнтним спуском і дозволяє моделі ефективно працювати в умовах обмеженої оперативної пам`яті.
- Перемішування (shuffle=True): Перед кожною епохою порядок даних випадковим чином змінюється. Це критично важливо для того, щоб модель не запам`ятовувала послідовність пакетів у часі, а вчилася узагальнювати їхні характеристики.

Результат виконання коду зберігається у змінну history, яка містить детальну статистику зміни метрик точності на кожному кроці. Як видно з логів виконання (Output комірки), значення функції втрат (Mean Absolute Error) стабільно знижувалося з кожною епохою, досягнувши фінального значення близько 0.008, що свідчить про успішну збіжність алгоритму та високу якість реконструкції нормального трафіку.

```
print("Навчання Autoencoder")

history = autoencoder.fit(X_train_scaled, X_train_scaled,
                          epochs=20,
                          batch_size=32,
                          shuffle=True,
                          verbose=1)

print("Модель Autoencoder навчена")
```

Рисунок 3.6 - Навчання моделі Autoencoder

Завершальним етапом програмної реалізації у середовищі Jupyter Notebook є валідація навченої моделі Автокодувальника на тестовій вибірці. Цей процес реалізовано у десятій комірці і складається з кількох алгоритмічних кроків, що перетворюють вихідні дані нейромережі (реконструйовані вектори) у бінарні вердикти (“Норма” або “Атака”).

Генерація реконструкцій та розрахунок похибки. Першим кроком алгоритм пропускає масштабований тестовий набір даних X_test_scaled через навчену нейромережу за допомогою методу predict. На виході отримується масив реконструйованих даних, який має ту саму розмірність, що і вхідний. Далі обчислюється міра відхилення між оригінальними та відновленими даними. Для цього, аналогічно до етапу навчання, використовується Середня Абсолютна Помилка (MAE). За допомогою бібліотеки NumPy (np.abs та np.mean з параметром axis=1) розраховується усереднене значення помилки для кожного окремого пакета (рядка таблиці). Цей вектор помилок має виступає в ролі “оцінки аномальності” (Anomaly Score).

Розрахунок адаптивного порогу (Thresholding). Критичним аспектом роботи будь-якої системи NIDS є визначення межі, після якої відхилення вважається атакою. У роботі реалізовано статистичний підхід до визначення

цього порогу. Для цього модель спочатку робить прогноз на тренувальній вибірці X_train_scaled (яка містить лише нормальний трафік) і розраховує помилки реконструкції для неї (train_mae). Поріг (threshold) визначається як 95-й перцентиль (квантиль 0.95) розподілу помилок на нормальному трафіку.

Це означає, що система налаштовується таким чином, щоб розпізнавати 95% навчального трафіку як “нормальний”, залишаючи 5% запасу на можливі статистичні викиди або шум. Такий підхід дозволяє адаптувати чутливість системи під конкретний профіль мережі без “жорсткого кодування” констант.

Бінарна класифікація та метрики. На основі розрахованого порогу відбувається фінальна класифікація тестових даних. Якщо помилка реконструкції конкретного пакета перевищує поріг, йому присвоюється мітка 1 (Атака), в іншому випадку - 0 (Норма). Отриманий вектор прогнозів y_pred_auto порівнюється з істинними мітками y_test_normalized для розрахунку метрик ефективності:

- Accuracy (Точність): Загальний відсоток правильних відповідей.
- Classification Report: Детальний звіт, що включає Precision (точність), Recall (повнота) та F1-score для кожного класу.
- Confusion Matrix (Матриця похибок): Візуалізація результатів у вигляді теплової карти (heatmap), що дозволяє миттєво оцінити кількість пропущених атак (False Negatives) та хибних спрацьовувань (False Positives).

Як видно з результатів виконання коду, модель Автокодувальника демонструє значно вищі показники F1-score для класу атак порівняно з Isolation Forest, що підтверджує гіпотезу про перевагу методів глибокого навчання для аналізу складних патернів мережевого трафіку.

```
reconstructions = autoencoder.predict(X_test_scaled)

mae = np.mean(np.abs(X_test_scaled - reconstructions), axis=1)

reconstructions_train = autoencoder.predict(X_train_scaled)
train_mae = np.mean(np.abs(X_train_scaled - reconstructions_train), axis=1)

threshold = np.quantile(train_mae, 0.95)
print(f"Обраний поріг (95-й перцентиль помилки MAE): {threshold}")

y_pred_auto = [1 if e > threshold else 0 for e in mae]

accuracy_auto = accuracy_score(y_test_normalized, y_pred_auto)
print(f"\nЗагальна точність Autoencoder: {accuracy_auto * 100:.2f}%\n")

print("Звіт по класифікації (Autoencoder):")
print(classification_report(y_test_normalized, y_pred_auto, target_names=['Норма (0)', 'Атака (1)']))

print("Матриця похибок (Autoencoder):")
cm_auto = confusion_matrix(y_test_normalized, y_pred_auto)
sns.heatmap(cm_auto, annot=True, fmt='d', cmap='Blues', xticklabels=['Норма', 'Атака'], yticklabels=['Норма', 'Атака'])
plt.xlabel('Прогноз')
plt.ylabel('Реальний')
plt.show()
```

Рисунок 3.7 - Оцінка ефективності системи

Після успішного навчання та валідації моделей, завершальним етапом роботи в середовищі Jupyter Notebook є підготовка системи до розгортання в режимі реального часу. Реалізовано процедуру серіалізації (збереження) навчених об`єктів для їх подальшого використання у зовнішньому скрипті main.py.

За допомогою бібліотеки Joblib зберігається екземпляр scaler, який містить параметри нормалізації (мінімальні та максимальні значення для кожної з 78 ознак), отримані з тренувального набору. Це критично важливо для забезпечення ідентичності перетворень даних на етапі експлуатації. Також зберігається модель Isolation Forest у форматі .pkl. Для нейронної мережі Autoencoder використовується власний метод save бібліотеки Keras, який експортує архітектуру та ваги моделі у формат .h5. Цей крок перетворює результати експериментального навчання на відчужувані програмні компоненти, готові до інтеграції в систему моніторингу.

```
import joblib
from tensorflow.keras.models import load_model

joblib.dump(model_iso, 'isolation_forest_model.pkl')

autoencoder.save('autoencoder_model.h5')

joblib.dump(scaler, 'scaler.pkl')

print("Всі моделі та Scaler збережені")
```

Рисунок 3.8 - Збереження навчених об`єктів

Для того щоб переконатися, що система коректно обробляє окремі вектори даних поза межами великих датафреймів, програмний код витягує по одному еталонному рядку для кожного класу. З масиву X_train (який гарантовано містить нормальний трафік) витягується перший запис, що слугуватиме “золотим стандартом” норми. З масиву X_test витягується запис, який відповідає атаці (фільтрація за умовою y_test != `BENIGN`). Ці вектори конвертуються у стандартні списки Python та виводяться на екран. Отримані числові послідовності надалі використовуються в скрипті main.py для імітації вхідного потоку даних та перевірки реакції системи на відомі загрози.

```
normal_row = X_train.iloc[0].values.tolist()

attack_row = X_test[y_test != 'BENIGN'].iloc[0].values.tolist()

print("----- РЯДОК НОРМИ -----")
print(normal_row)
print("\n----- РЯДОК АТАКИ -----")
print(attack_row)
```

Рисунок 3.9 – Вилучення даних норми та атаки

Фінальна комірка присвячена глибокому статистичному аналізу розподілу помилок реконструкції, що є математичним обґрунтуванням ефективності запропонованого методу. Код розділяє масив помилок mae на дві підгрупи: помилки для нормального трафіку та помилки для атак. За допомогою методу describe() розраховуються описові статистики (середнє, стандартне відхилення, квартилі) для кожної групи.

Результати цього аналізу, наведені у висновку програми, наочно демонструють фундаментальну різницю у сприйнятті трафіку автокодувальником: середня помилка реконструкції для легітимних пакетів є на порядок нижчою, ніж для аномальних. Це статистичне підтвердження є головним доказом того, що нейронна мережа успішно вивчила приховані закономірності нормальної поведінки мережі та здатна дискримінувати аномалії на основі величини похибки відновлення. Отримане значення порогу (threshold) є математично обґрунтованою межею, яка розділяє ці два статистичні розподіли з заданим рівнем довіри.

```
mae_series = pd.Series(mae)

y_test_reset = y_test_normalized.reset_index(drop=True)

normal_errors = mae_series[y_test_reset == 0]
attack_errors = mae_series[y_test_reset == 1]

print("--- Статистика помилок Autoencoder (MAE) ---")
print("\nПОМИЛКИ НОРМАЛЬНОГО ТРАФІКУ (Benign):")
print(normal_errors.describe())

print("\nПОМИЛКИ АТАК (Attack):")
print(attack_errors.describe())

print(f"\nПорiг (Threshold): {threshold}")
```

Рисунок 3.10 – Статистичний аналіз розподілу помилок

3.2 Реалізація підсистеми моніторингу та детектування атак у реальному часі

Розроблений скрипт main.py виконує роль демонстраційного прототипу системи виявлення вторгнень. Його основна мета - продемонструвати здатність попередньо навчених моделей (Isolation Forest та Autoencoder) коректно класифікувати нові, раніше невідомі вектори мережевого трафіку.

Алгоритм роботи скрипта можна розділити на три логічні етапи: ініціалізація, логіка детектування та симуляція вхідного потоку.

На початку роботи програма виконує підключення необхідних бібліотек: joblib для роботи з об`єктами Scikit-learn, tensorflow для завантаження нейронної мережі та numpy для матричних операцій.

Ключовим етапом є завантаження збережених компонентів системи («артефактів навчання») в оперативну пам`ять:

1. Об`єкт нормалізації (scaler.pkl): Завантажується екземпляр класу MinMaxScaler. Це критично важливо для забезпечення консистентності даних: нові вектори трафіку повинні бути нормалізовані за тими ж правилами (мінімумами та максимумами), що й навчальна вибірка.

2. Статистична модель (isolation_forest_model.pkl): Завантажується навчена модель Isolation Forest, готова до виконання методу predict.

3. Нейромережева модель (autoencoder_model.h5): Завантажується архітектура та ваги глибокого автокодувальника.

Також у цьому блоці встановлюється поріг чутливості системи (THRESHOLD_AUTOENCODER), значення якого було отримано емпіричним

шляхом на етапі навчання (як 95-й перцентиль помилки на нормальному трафіку).

У коді реалізовано дві окремі функції для перевірки трафіку різними методами, що дозволяє порівняти їх ефективність.

Функція check_with_isolation_forest:

Ця функція реалізує підхід на основі дерев рішень.

1. Підготовка даних: Вхідний список значень перетворюється на двовимірний масив NumPy (reshape(1, -1)), оскільки модель очікує на вхід матрицю.

2. Масштабування: Застосовується метод transform завантаженого скейлера.

3. Прогноз: Метод predict повертає значення -1 (аномалія) або 1 (норма). На основі цього коду формується текстовий вердикт (“ТРИВОГА” або “Норма”).

Функція check_with_autoencoder:

Ця функція реалізує підхід на основі глибокого навчання (реконструкції даних).

1. Нормалізація: Аналогічно попередній функції, вхідний вектор приводиться до діапазону [0, 1].

2. Реконструкція: Нормалізований вектор подається на вхід автокодувальника. Мережа намагається стиснути дані та відновити їх на виході.

3. Розрахунок похибки: Обчислюється середньоквадратична помилка (MSA) між вхідним вектором та його реконструкцією

4. Прийняття рішення: Отримане значення помилки порівнюється із заданим порогом (THRESHOLD). Якщо помилка перевищує поріг, це свідчить про те, що модель не змогла впізнати патерн трафіку, отже, він є аномальним.

Блок if __name__ == “__main__”: імітує надходження пакетів з мережі.

Для демонстрації використовуються два статично задані вектори ознак (кожен містить 78 числових значень), отримані з реального датасету CIC-IDS 2017:

1. normal_packet: Вектор, що відповідає легітимному трафіку (отриманий з файлу понеділка). Очікується, що система класифікує його як “Норма”.

2. attack_packet: Вектор, що відповідає реальній атаці (Brute Force або DDoS). Очікується, що система класифікує його як “ТРИВОГА”.

Програма послідовно передає ці вектори обом моделям та виводить результати в консоль, що дозволяє наочно перевірити працездатність розробленої системи захисту без необхідності розгортання повномасштабної мережевої інфраструктури.

```
Моделі готові до роботи
--- Перевірка 'Нормального' пакета ---
ISOLATION FOREST:       Норма.
AUTOENCODER:            Норма. (Помилка: 0.013065)

--- Перевірка 'Аномального' пакета ---
ISOLATION FOREST:       ТРИВОГА! Аномалія виявлена!
AUTOENCODER:            ТРИВОГА! Аномалія виявлена! (Помилка: 0.029588)
```

Рисунок 3.11 – Демонстрація розпізнавання трафіку

Висновки до розділу:

У третьому розділі магістерської роботи здійснено програмну реалізацію та проведено експериментальне дослідження інтелектуальної системи виявлення мережевих аномалій ML-NADS. Розроблений програмний комплекс структурно складається з двох функціональних компонентів: модуля офлайн-навчання та валідації моделей (program), а також модуля моніторингу та детектування загроз у реальному часі (main), реалізованих мовою Python із використанням спеціалізованих бібліотек Scikit-learn, TensorFlow/Keras, Pandas, NumPy.

Ключовим досягненням етапу підготовки даних стало успішне впровадження стратегії навчання «One-Class Learning», яка дозволила сформувати високоякісну навчальну вибірку на основі «чистого» трафіку з датасету CIC-IDS 2017. Для забезпечення коректної роботи нейронної мережі було реалізовано алгоритми попередньої обробки, зокрема нормалізацію даних методом MinMax Scaling, що приводить усі ознаки до єдиного діапазону [0, 1]. Основну увагу в роботі приділено розробці та оптимізації глибокого автокодувальника, де експериментальним шляхом було встановлено, що використання функції втрат MAE (Mean Absolute Error) у поєднанні із сигмоїдальною активацією вихідного шару забезпечує найкращу стійкість до шумів та дозволяє чітко ідентифікувати аномалії за значним зростанням помилки реконструкції.

Для забезпечення функціонування системи в умовах реальної мережі створено модуль детектування (main). Цей модуль інтегрує попередньо навчені моделі та здійснює класифікацію трафіку «на льоту», порівнюючи поточну помилку реконструкції з розрахованим адаптивним порогом. Проведена експериментальна верифікація та порівняльний аналіз підтвердили перевагу розробленої моделі Autoencoder над статистичним методом Isolation Forest,

продемонструвавши вищу точність виявлення атак та кращу здатність до узагальнення нормальної поведінки. Фінальне тестування прототипу довело його здатність коректно розрізняти легітимний веб-трафік та інтенсивну мережеву активність, характерну для кібератак, що свідчить про повну працездатність запропонованого програмного рішення.

ВИСНОВКИ

У ході дослідження було встановлено, що традиційні засоби захисту, такі як міжмережеві екрани та сигнатурні системи виявлення вторгнень, втрачають свою ефективність в умовах стрімкого розвитку кіберзагроз. Їхня залежність від баз відомих сигнатур робить їх вразливими до новітніх векторів атак, таких як загрози нульового дня (Zero-day), APT-атаки та поліморфні віруси. На основі цього аналізу було обґрунтовано доцільність переходу до методів машинного навчання, зокрема навчання без учителя (Unsupervised Learning). Такий підхід дозволяє ідентифікувати загрози не за їхнім цифровим «підписом», а за статистичним відхиленням від моделі нормальної поведінки мережі.

Для практичної реалізації системи було обрано та науково обґрунтовано технологічний стек на базі мови програмування Python з використанням бібліотек Scikit-learn, TensorFlow/Keras, Pandas та NumPy. Це забезпечило високу гнучкість розробки та необхідну продуктивність обчислень. Центральним елементом системи стала спроєктована архітектура глибокої нейронної мережі - Автокодувальника (Autoencoder). Для порівняльного аналізу також було реалізовано статистичний метод Isolation Forest. Важливим науковим результатом став вибір методу попередньої обробки даних (MinMax Scaling) та функції втрат (Mean Absolute Error - MAE), що дозволило значно підвищити стійкість нейромережі до статистичних викидів у вхідних даних та покращити якість навчання.

Навчання моделей проводилося виключно на «чистому» трафіку з датасету CIC-IDS 2017, що дозволило сформувати еталонний профіль нормальної роботи мережі.

Експериментальна перевірка підтвердила ефективність запропонованого підходу. Порівняльний аналіз показав перевагу нейромережевої моделі:

Autoencoder продемонстрував кращу здатність до узагальнення та вищий показник F1-score у порівнянні з Isolation Forest. Фінальні випробування розробленого модуля моніторингу довели його здатність аналізувати живий потік даних та генерувати сповіщення про аномалії при виявленні відхилень, що перевищують розрахований адаптивний поріг. Таким чином, розроблена система є дієвим інструментом для виявлення як відомих, так і невідомих раніше кібератак, а запропонована архітектура має значний потенціал для подальшого розвитку адаптивних систем кібербезпеки.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. IT Ukraine Association. Ukraine Cybersec Market Review. URL: https://itukraine.org.ua/files/Ukraine-Cybersec-Market-Review.pdf (дата звернення: 01.12.2025).

2. Державна служба спеціального зв'язку та захисту інформації України. Аналітичний звіт про кіберагресію росії проти України за 2023 рік. URL: https://scpc.gov.ua/api/files/9c21855d-74da-45d1-90f9-5d4f6795996a (дата звернення: 01.12.2025).

3. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, 2015. 784 p.

4. Chappell D. Understanding Active Directory. O`Reilly Media, 2017. 304 p.

5. RFC 5246 – The Transport Layer Security (TLS) Protocol. URL: https://datatracker.ietf.org/doc/html/rfc5246 (дата звернення: 19.08.2025).

6. Encryption Consulting. What is Cryptography? URL: https://www.encryptionconsulting.com/education-center/what-is-cryptography/ (дата звернення: 01.12.2025).

7. Павленко О. В., Кравченко М. П. Захист комп`ютерних мереж від кіберзагроз: сучасні методи та технології. Київ : Наукова думка, 2018. 368 с.

8. Patel A., Taghavi M. Intrusion Detection Systems and Prevention Techniques. Elsevier, 2016. 368 p.

9. Northcutt S., Novak J. Network Intrusion Detection. New Riders, 2016. 576 p.

10. Cybersecurity and Infrastructure Security Agency (CISA). URL: https://www.cisa.gov/ (дата звернення: 19.08.2025).

11. Computer Security Incident Handling Guide. NIST Special Publication 800-61. URL: https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final (дата звернення: 19.08.2025).

12. SoftwarePair. Does Antivirus Protect Against Modern Ransomware? URL: https://softwarepair.com/does-antivirus-protect-against-modern-ransomware/ (дата звернення: 01.12.2025).

13. Radvanovsky R., Brodsky J. Cybersecurity Policies and Strategies for Cyberwarfare Prevention. CRC Press, 2016. 328 p.

14. Mirkovic J., Reiher P. Understanding DDoS Attacks. IEEE Communications Magazine. 2018. Vol. 56, № 5. P. 17–23.

15. Indusface. What is a DDoS Attack? URL: https://www.indusface.com/learning/what-is-a-ddos-attack/ (дата звернення: 01.12.2025).

16. RFC 4301 – Security Architecture for the Internet Protocol. URL: https://datatracker.ietf.org/doc/html/rfc4301 (дата звернення: 19.08.2025).

17. System Solution. Data Backup and Recovery. URL: https://www.systemsolution.ca/2016/12/19/data-backup-and-recovery/ (дата звернення: 01.12.2025).

18. Метод і пристрій для динамічного контролю доступу: пат. US20250148064 США: МПК G06F 21/00. № US20250148064; заявл. 2024; опубл. 2025.

19. Багатофакторна мультимедійна біометрична автентифікація: пат. US20090116703A1 США: МПК G06K 9/00, G06F 21/32. № US20090116703A1; заявл. 2008; опубл. 07.05.2009.

20. Системи й методи виявлення загроз на основі поведінкового аналізу: пат. US20200186546A1 США: МПК H04L 29/06, G06F 21/55. № US20200186546A1; заявл. 2019; опубл. 11.06.2020.

21. Протасов В. І. Кіберзахист: методи і технології. Київ : Магістр, 2021. 512 с.

22. Моніторинг та запобігання автоматизованим кібератакам віддалених користувачів: пат. US11722510B2 США: МПК H04L 63/14. № US11722510B2; заявл. 2021; опубл. 01.08.2023.

23. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1600 p.

24. Visual Studio Code User Guide. URL: https://code.visualstudio.com/docs (дата звернення: 12.10.2025).

25. Array programming with NumPy / C. R. Harris et al. // Nature. 2020. Vol. 585. P. 357–362.

26. McKinney W. Python for Data Analysis : Data Wrangling with Pandas, NumPy, and IPython. 2nd ed. Sebastopol : O'Reilly Media, 2017. 544 p.

27. Scikit-learn: Machine Learning in Python / F. Pedregosa et al. // Journal of Machine Learning Research. 2011. Vol. 12. P. 2825–2830.

28. A System for Large-Scale Machine Learning / M. Abadi et al. // 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2016. P. 265–283.

29. Chollet F. Deep Learning with Python. Shelter Island : Manning Publications, 2017. 384 p.

30. Hunter J. D. Matplotlib: A 2D graphics environment // Computing in Science and Engineering. 2007. Vol. 9, no. 3. P. 90–95.

31. Waskom M. L. Seaborn: statistical data visualization // Journal of Open Source Software. 2021. Vol. 6, no. 60. P. 3021. URL: https://seaborn.pydata.org/ (дата звернення: 20.10.2025).

32. Маринич І. А., Тронь В. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 151 "Автоматизація

та комп`ютерно-інтегровані технології”. Кривий Ріг : Видавничий центр КНУ, 2022. 50с.

33. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

34. ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

35. ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація)

36. ДСТУ 3651.0-97 Метрологія. Одиниці фізичних величин. Основні одиниці фізичних величин Міжнародної системи одиниць. Основні положення, назви та позначення Київ, Держстандарт України, 1998. 27 с. (Інформація та документація).

ДОДАТОК

Лістинг коду скрипту системи виявлення вторгнень

program.ipynb | main.py 1 | Monday-WorkingHours.pcap_ISCX.csv

main.py > check_with_autoencoder

```python
import joblib
from tensorflow.keras.models import load_model
import numpy as np
import warnings

warnings.filterwarnings('ignore', category=UserWarning)

print("Завантаження моделей")
scaler = joblib.load('scaler.pkl')
model_iso = joblib.load('isolation_forest_model.pkl')
autoencoder = load_model('autoencoder_model.h5')

THRESHOLD_AUTOENCODER = 0.025960823566442113

print("Моделі готові до роботи")

def check_with_isolation_forest(packet_data):
    feature_vector = np.array(packet_data).reshape(1, -1)


    feature_vector_scaled = scaler.transform(feature_vector)


    prediction = model_iso.predict(feature_vector_scaled)

    if prediction[0] == -1:
        return "ISOLATION FOREST: \tТРИВОГА! Аномалія виявлена!"
    else:
        return "ISOLATION FOREST: \tНорма."
```

```python

def check_with_autoencoder(packet_data):
    feature_vector = np.array(packet_data).reshape(1, -1)

    feature_vector_scaled = scaler.transform(feature_vector)

    reconstruction = autoencoder.predict(feature_vector_scaled, verbose=0)
    mae = np.mean(np.abs(feature_vector_scaled - reconstruction), axis=1)

    if mae[0] > THRESHOLD_AUTOENCODER:
        return f"AUTOENCODER: \t\tТРИВОГА! Аномалія виявлена! (Помилка: {mae[0]:.6f})"
    else:
        return f"AUTOENCODER: \t\tНорма. (Помилка: {mae[0]:.6f})"

if __name__ == "__main__":

    normal_packet = [49188.0, 4.0, 2.0, 0.0, 12.0, 0.0, 6.0, 6.0, 6.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3000000.0, 50

    attack_packet = [389,113473706,68,40,11364,12718,403,0,167.1176471,171.9194127,1139,126,317.95,208.261294

    print("--- Перевірка 'Нормального' пакета ---")
    print(check_with_isolation_forest(normal_packet))
    print(check_with_autoencoder(normal_packet))

    print("\n--- Перевірка 'Аномального' пакета ---")
    print(check_with_isolation_forest(attack_packet))
    print(check_with_autoencoder(attack_packet))
```

```
Моделі готові до роботи
--- Перевірка 'Нормального' пакета ---
ISOLATION FOREST:       Норма.
AUTOENCODER:            Норма. (Помилка: 0.013065)

--- Перевірка 'Аномального' пакета ---
ISOLATION FOREST:       ТРИВОГА! Аномалія виявлена!
AUTOENCODER:            ТРИВОГА! Аномалія виявлена! (Помилка: 0.029588)
```