Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав. кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022 р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Розробка експертної системи нейромережевого детектору зловмисного трафіку»

оцінка « »

Голова ЕК

Доценко С.І. _____

Керівник _проф., д.т.н. Єсін В.І._

Рецензент проф., д.т.н. Доля Г.М.

Виконавець : студентка групи КБ-

_____Рогоза П. В.

Харків – 2022

РЕФЕРАТ

Пояснювальна записка містить 77 сторінок, 68 рисунків, 9 формул, 4 таблиці, 23 джерела, 1 додаток.

У роботі розроблено експертну систему нейромережевого детектору зловмисного трафіку веб-серверів, а також рекомендації для її застосування. Експертна система складається з нейронної мережі, бази даних та інтерфейсу користувача. Основна мета її застосування – підвищення захищеності веб-ресурсів від кібератак (типу SQLi, XSS, SSI, CRLF тощо) шляхом забезпечення швидкої обізнаності фахівців інформаційної безпеки про наявність атаки.

У роботі використані наступні методи дослідження: теорія баз даних, нейронних мереж, алгоритмів, методи аналізу метрик якості та великих масивів даних.

Проект спрямований на вирішення наступних задач: визначення і аналіз головних загроз для веб-ресурсів на основі OWASP та CWE, формування вимог до експертної системи, обґрунтування вибору архітектури моделі нейронної мережі, розробка бази даних і забезпечення її захисту, тестування та оцінка якості експертної системи, розробка рекомендації для покращення інформаційної безпеки веб-ресурсів.

Результатами дослідження є запропонована нейронна експертна система з її окремими компонентами: захищеною базою даних, навченою та протестованою моделлю нейронної мережі, яка забезпечує 98% коректного детектування та класифікації ін’єкцій у веб-ресурси, а також менше 5% виникнення помилок першого та другого роду у відповідності до використаного набору даних.

Ключові слова: ЕКСПЕРТНА СИСТЕМА, ЗАХИСТ WEB-ДОДАТКІВ, КІБЕРАТАКА, НЕЙРОННА МЕРЕЖА.

ABSTRACT

The explanatory note contains 77 pages, 68 figures, 9 formulas, 4 tables, 23 sources, 1 appendix.

In this work, an expert system of web resources malicious traffic detector based on the neural network was developed, as well as recommendations for its application. The expert system consists of a neural network, a database and a user interface. The main purpose of its usage is to increase the security of web resources against cyberattacks (such as SQLi, XSS, SSI, CRLF etc.) by ensuring that information security specialists are quickly aware of the attack presence.

The following research methods were used: the databases theory, neural networks, algorithms theory, and the methods for analyzing quality metrics and large data sets.

The project is aimed at solving the following problems: identifying and analyzing the main threats to web resources based on OWASP and CWE, forming requirements for an expert system, justifying the choice of the neural model architecture, database development and ensuring the protection, testing and evaluation of the expert system quality, development of recommendations for the web resources information security improvement.

The results of the research are the proposed neural expert system with individual components: a secure database, a trained and tested neural network model that provides 98% correct detection and classification of injections into web resources, as well as less than 5% the first and second kinds of errors occurrence according to the used data set.

Keywords: CYBER ATTACK, EXPERT SYSTEM, NEURAL NETWORK, WEB APPLICATION PROTECTION.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

AI	–	Artificial Intelligence (Штучний Інтелект)
CSRF	–	Cross-Site Request Forgery
CWE	–	Common Weakness Enumeration (Загальний перелік слабкості)
NIST	–	National Institute of Standards and Technology (Національний інститут стандартів і технологій США)
OWASP	–	Open Web Application Security Project (Відкритий проект з безпеки веб-застосунків)
SQL	–	Structured Query Language (Мова Структурованих Запитів)
XEE	–	XML External Entities Injection
XSS	–	Cross Site Scripting
ІзОД	–	Інформація з обмеженим доступом
ІБ	–	Інформаційна безпека
ІТС	–	Інформаційно-Телекомунікаційна Система
НМ	–	Нейронна мережа
ПЗ	–	Програмне Забезпечення

ВСТУП

Сучасний світ інформаційних технологій надає нам широкий спектр веб-застосунків. Для їх коректної та стабільної роботи, звісно, існує постійна необхідність у надійному захисті веб-ресурсів та конфіденційної інформації, яка на них зберігається. Зі збільшенням числа кібератак зростають також критичні наслідки від них як для організацій, так і для приватних осіб.

Проблема збільшення кількості кібератак у світі потребує своєчасного інформування фахівців інформаційної безпеки (ІБ) про поточну кібератаку. На сучасному етапі розвитку є кілька підходів на вирішення цієї проблеми, зокрема з використанням інтелектуальних систем. Однак, нині засоби захисту, які засновані на штучних нейронних мережах (НМ), досі надають широкий потенціал для різних наукових випробувань, зокрема для захисту веб-ресурсів, і не є повноцінно дослідженими [1]. Тому, враховуючи те, що застосування технології нейронних мереж має багато переваг і можливостей, що дозволяють вирішити цю проблему, дана тема дослідження є актуальною.

У роботі запропоновано можливе рішення цієї проблеми – розробка нейронної експертної системи детектору зловмисного трафіку веб-серверів, а також надано рекомендації для її застосування. Створення автономної експертної системи дозволить знизити навантаження на спеціалістів з безпеки, підвищити швидкість аналізу трафіку та виключити суб’єктивну складову у прийнятті рішення, на відміну від експертної думки. Таким чином, можливе ефективне зменшення втрат підприємств (економічних, інформаційних, репутаційних), а також підвищення захищеності ресурсів від кібератак.

Новизна роботи полягає у розробці експертної системи, яка здатна своєчасно детектувати та класифікувати загрози на основі НМ, та яка підвищує комплексне сповіщення фахівців ІБ про проведення атаки.

У першому розділі розглядаються поняття експертної системи та нейронної мережі як два представника Штучного Інтелекту. На основі аналізу цих

технологічних рішень, вилучення їх переваг та подібностей, вони були пов'язані у єдину систему, яка зберігає особливості кожного окремого рішення. Для нейронної експертної системи висунуті функціональні і нефункціональні вимоги, обмеження предметної області та очікувані результати роботи.

У другому розділі зроблено концептуальне і логічне проектування бази даних для нейронної мережі, а також розроблені механізми захисту бази даних відповідно до попередньо поставлених вимог.

У третьому розділі проводилась практична частина роботи – програмна реалізація та дослідження нейронної мережі. Здійснені заходи поєднання компонентів системи воєдино завдяки розробленому API з'єднання з БД. Головним результатом розділу є імплементована експертна система, працююча відповідно до виставлених вимог, з відповідними показниками нейронної мережі. Зокрема, були порівняні метрики якості чотирьох різних архітектур класифікаторів після їх навчання на однаковому наборі даних: Random Forest, K-Neighbors, Decision Tree, Convolutional Neural Network. Кількісними метриками (показниками точності класифікації) доведено, що найбільш оптимальною архітектурою є згорткова НМ, яка має загальну точність детектування 99%. Інші мережі також надали прийнятні результати (між 96-98% точності), і тому були збережені на випадок непередбачуваних результатів погіршення точності кожної архітектури на реальних мережевих даних.

Окремо сформовані рекомендації зі застосування розробленої системи та показано, у якому разі захист веб-ресурсів буде наданий якісно та професіонально.

1 НЕОБХІДНІСТЬ ПОБУДОВИ ЕКСПЕРТНОЇ СИСТЕМИ

1.1 Основні поняття, призначення та застосування експертних систем

Експертна система (ЕС) – це комп'ютерна система штучного інтелекту, призначена для вирішення складних проблем (прогнозування, контролювання, планування, управління, навчання тощо) і надання можливості автономно приймати такі рішення, які здатна прийняти і людина-експерт [2].

ЕС надає рішення проблеми, витягуючи знання зі своєї бази знань, послідовно застосовуючи відповідний до задачі набір правил і висновків відповідно до запитів користувачів. Одним із найпростіших прикладів функціонування окремої експертної системи є пропозиція змінити запит у вікні пошуку Google, який містить орфографічні помилки з моменту введення тексту людиною.

Експертна система не використовується для повноцінної заміни експертів-людей; замість цього вона використовується, щоб допомогти людині прийняти складне рішення. Ці системи не мають людських можливостей абстрактно мислити і працювати на основі обширної бази знань конкретної області (медицини, науки тощо). Продуктивність експертної системи базується кількості на знаннях експерта, які зберігаються в базі знань. Чим більше знань зберігається в базі, тим більше ця система покращує свою продуктивність.

Експертна система в основному складається з трьох компонентів (див. рис. 1.1):

1) Інтерфейс користувача (англ. User Interface). Найважливіша частина ЕС, оскільки вона отримує запити користувача у зрозумілому форматі та доставляє його до машини логічного виведення. Інтерфейс показує результат (пораду) для користувача. Простіше кажучи, це комунікаційний міст, який з'єднує користувачів і експертну систему.
2) Машина логічного виведення (англ. Interface Engine). Також відома як машина висновування, вона працює як мозок експертної системи. Interface Engine містить умови та правила вирішення конкретного питання. Її знання надходять із бази знань

для надання міркувань щодо збереженої інформації. Існує два типи машини логічного виведення:

– Механізм детермінованого висновку: Висновки, зроблені за допомогою цього типу механізму логічного висновку, вважаються істинними. Він заснований на фактах і правилах.
– Механізм ймовірнісного висновку: цей тип механізму логічного висновку містить невизначеність у висновках і ґрунтується на ймовірності.

Крім того, машина використовує стратегії прямого та зворотного зв'язування (англ. forward and backward chaining), щоб завершити свою відповідь (пораду людині) на дану проблему.

– Пряме зв'язування: у цій стратегії Interface Engine спочатку розглядає певні правила та факти, перш ніж поділитися висновком. Алгоритм прямого зв'язування починається з відомих фактів, запускає всі правила, передумови яких виконуються, і додає свій висновок до відомих фактів. Цей процес повторюється, доки проблема не буде вирішена.
– Зворотнє зв'язування: у цьому випадку Interface Engine намагається знайти можливі умови, які могли статися в минулому, щоб знайти остаточну пораду. Алгоритм зворотного зв'язування – це форма міркування, яка починається з цілі та працює у зворотному напрямку, з'єднуючи правила для пошуку відомих фактів, які підтверджують ціль. Він застосовний, коли необхідне міркування для конкретної ситуації.

3) База знань (англ. Knowledge Base). Зберігає всю інформацію, частіше за все, передану від експертів, про проблему, що розглядається. Іншими словами, це сховище фактів, яке містить знання з різних джерел. Компоненти бази знань подаються як:

– Фактичні знання: ці знання прийняті експертами в галузі та базуються на фактах і правилах.
– Евристичні знання: ця інформація надходить через досвід або практику. Крім того, до цих знань входять здатність припускати, оцінювати та практичний досвід.

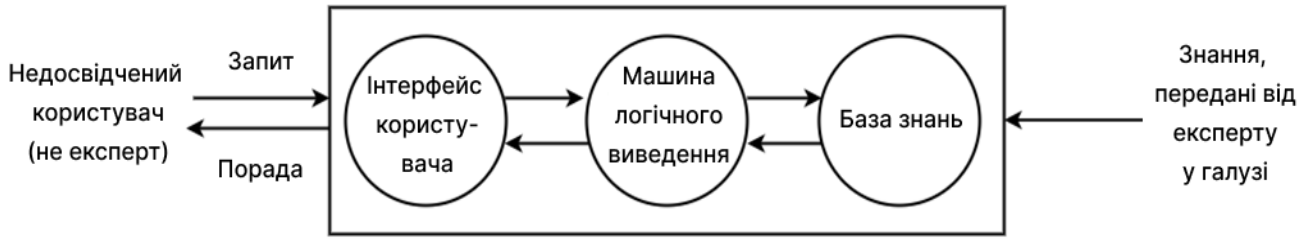


Рисунок 1.1 – Структура класичної експертної системи

Перш ніж використовувати будь-яку технологію, потрібно мати уявлення про те, навіщо вона потрібна, які її переваги та обмеження, а отже, необхідно зробити загальний аналіз підходу, який заснований на експертних системах. Потреба у ЕС обумовлена наступними факторами:

1) Немає обмежень пам'яті. ЕС може зберігати необхідну кількість даних і запам'ятовувати їх під час застосування. Але для людей-експертів існують певні обмеження щодо запам'ятовування всього в будь-який час.
2) Висока ефективність. Якщо база знань оновлюється коректними знаннями, вона забезпечує високоефективний результат, який може бути неможливим для людини.
3) Експертиза в домені. У кожній області є багато експертів-людей, і всі вони мають різні навички, різний досвід і різні навички, тому отримати остаточний результат для запиту нелегко. Але якщо ми помістимо знання, отримані від експертів-людей, в експертну систему, то вона забезпечить ефективний результат шляхом змішування всіх фактів і знань
4) Вплив емоцій. На ці системи не впливають такі людські емоції, як втома, гнів, депресія, тривога тощо. Тому продуктивність залишається незмінною.
5) Високий рівень безпеки. Ці системи забезпечують високий рівень безпеки для вирішення будь-яких запитів.
6) ЕС розглядає всі факти. Щоб відповісти на будь-який запит, вона перевіряє та розглядає всі наявні факти та надає відповідний результат. Але не виключено, що людина-експерт з будь-яких причин може не врахувати деякі факти.
7) Регулярні оновлення покращують продуктивність. Якщо є проблема в результатах, наданих експертними системами, ми можемо покращити продуктивність системи, оновивши базу знань.

- Хоча сьогодні рідше можна зустріти частини програмного забезпечення, які явно називаються «експертними системами», ідеї та реалізації представлення знань і висновків все ще широко використовуються. Приклади існуючих ЕС надають змогу зрозуміти та оцінити реальні масштаби проблем, які вирішуються. Окрім цього, якщо існують актуальні дослідження та готові рішення зі застосування ЕС, це є підтвердженням того, що ця технологія є затребуваною та корисною у сучасному світі. Відомі наступні застосування експертних систем [3]:DENDRAL: один із перших проектів штучного інтелекту, створений у 1965 році як експертна система хімічного аналізу. Його використовували в органічній хімії для виявлення невідомих органічних молекул за допомогою їх мас-спектрів і бази знань з хімії.
- MYCIN: одна з перших експертних систем зворотного зв'язування, яка була розроблена для пошуку бактерій, що викликають інфекції, такі як бактеріємія та менінгіт. ЕС також використовували для рекомендації антибіотиків і діагностики захворювань згортання крові.
- PXDES: Ця система є одним із прикладів експертних систем, які беруть участь у виявленні раку легень, його типу та стадії. Для визначення захворювання робиться знімок верхньої частини тіла, який виглядає як тінь. Ця тінь визначає тип і ступінь пошкодження.
- ROSS (від компанії IBM Watson) використовує обробку природної мови для пошуку та надання юридичної інформації, від цитат до повних юридичних матеріалів. Завдяки фінансовій підтримці міжнародної юридичної фірми Dentons ROSS має партнерські стосунки з великими установами, включаючи Latham & Watkins і Vanderbilt Law.
- FINEVA: це багатокритеріальна система підтримки прийняття рішень на основі знань для оцінки ефективності та життєздатності нової компанії. Фінансовий аналіз фірм передбачає виявлення сильних і слабких сторін фірм, головним чином за допомогою процедур судження щодо якісної оцінки та інтерпретації фінансових показників.

1.2 Нейронні мережі у експертних системах

Нейронна мережа (модель) – це метод штучного інтелекту, у якому комп'ютерні системи обробляють, аналізують, класифікують дані подібно спрощеній нервовій системі мозку людини. НМ працює завдяки великої кількості взаємопов'язаних абстрактних нейронів. Вони є блоками обробки даних, розташованими пошарово у відповідності до конкретної архітектури. У класичному варіанті завжди присутні вхідний шар даних, прихований шар математичних перетворень даних та вихідний шар. Основна послідовність дій одного нейрона наступна:

- прийом сигналів від попередніх елементів мережі;
- комбінування вхідних сигналів;
- обчислення вихідного сигналу;
- передача вихідного сигналу такими елементами нейронної мережі.

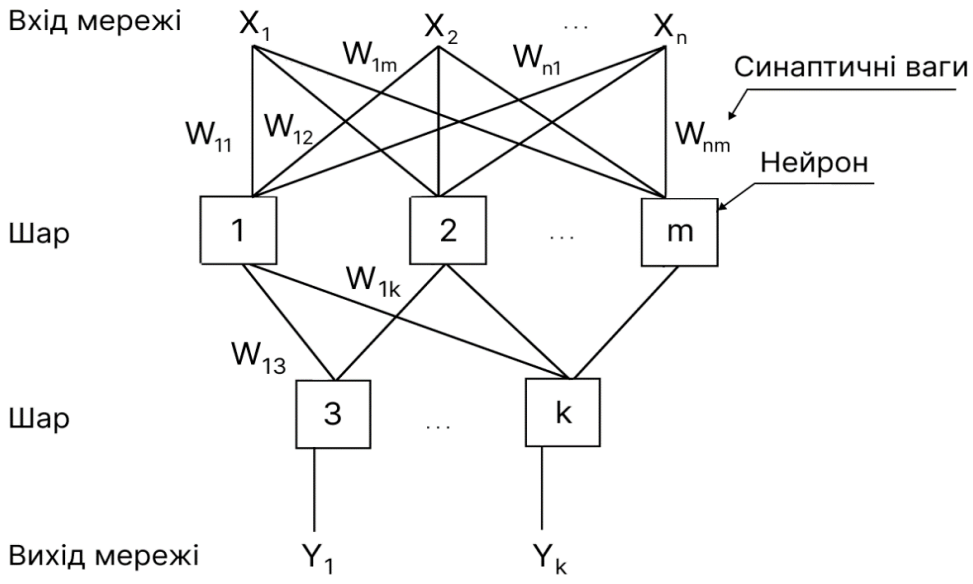


Рисунок 1.2 – Структура нейронної мережі

Між собою нейрони можуть бути з'єднані зовсім різним чином, що визначається структурою конкретної мережі.

Застосування нейронних мереж здійснюється поетапно:

1) постановка завдання – формування мети застосування НМ

2) навчання НМ – це заходи надання НМ попередньо оброблених навчальних даних, де вже є відомі результати вирішення задачі без нейромережі;
3) експлуатація мережі – пред'явлення мережі деяких нових невідомих ситуацій, які або розпізнаються або ні.

Навчання нейронних мереж є послідовним процесом зміни синаптичних ваг у початковій моделі, що відображають силу збудження зв'язків між нейронами.

Програмі нейронних мереж спочатку дається набір даних, який складається з багатьох змінних, які пов’язані з великою кількістю випадків чи результатів, у яких результати відомі. Початкова модель НМ, як правило, виконує велику кількість контрольованих навчальних завдань, будуючи стратегію прийняття рішень з того набору даних, де правильна відповідь надається заздалегідь. За кожне таке завдання початкова модель налаштовує показники ваг зв’язків між нейронами, підвищуючи точність своїх прогнозів. Для налаштування модель порівнює початкові результати з наданою правильною відповіддю або цільовим показником та фіксує певні кореляції. Цей процес програма нейронних мереж повторює багато разів, прагнучи покращити прогнозуючу здатність при налагодженні моделі. У результаті відбувається еволюція початкової моделі у навчену НМ, яка готова володіє достатнім досвідом для майбутніх передбачень.

Можна виділити наступні переваги застосування нейронних мереж [4]:

– Універсальність. Велика частина НМ здатна аналізувати навіть недосконалі вхідні дані – неповні, недостатні, занадто комплексні, які включають велику кількість параметрів. На всю вихідну генерацію не впливає пошкодження одного чи кількох нейронів, що робить нейронні мережі водночас відмовостійкими.

– Відносна простота. Побудова НМ за допомогою сучасних програмних засобів не є складною та не займає багато часу. Окрім цього, розробнику не обов’язково повноцінно розуміти внутрішнє функціонування штучних нейронів.

– Вихідні дані або результати роботи нейронної мережі не обмежуються кількістю вхідних даних.

– Нейронна мережа володіє певною властивістю паралелізації, тому що усі нейрони водночас включені у обробку даних.

Недоліки використовування нейронних мереж для вирішення задач, які потрібно врахувати при їх практичному застосуванні:

– Проблема вибору архітектури мережі. На відміну від необов'язковості знання про те, як функціонує НМ, розробнику необхідно успішно підібрати архітектуру шарів (яких існує велика кількість) для певного завдання. До того ж, стандартних архітектурних рішень може зовсім не існувати.

– Водночас з властивістю відносної простоти, існує також проблема інтерпретації результатів роботи, оскільки складні внутрішні механізми НМ залишаються "чорним ящиком". У ситуації, коли необхідно пояснити, на чому ґрунтуються перед-бачення моделі, часто це зробити неможливо.

Незважаючи на те, що НМ здатні вирішити практично будь-які завдання, у багатьох випадках їх застосування не є доцільним з точок зору: економічних, програмно-архітектурних, масштабування проекту, здобуття надто великої кількості реальних даних для навчання та тестування моделі. Для вирішення проблем іноді більш ефективним виявляється застосування класичних математичних моделей [5].

У той же час сучасні нейронні мережі продовжують відмінно впорюватись з проблемами класифікації, прогнозування, кодування і декодування інформації, а також мінімізують суб’єктивну та упереджену складову оцінки, що задовольняє поточній задачі детектування (прогнозування та класифікації) кіберзагроз у HTTP-трафіку. Окрім цього, підходи машинного навчання до програм кібербезпеки пропонують розумну можливість ідентифікувати нові варіанти зловмисного програмного забезпечення та атаки нульового дня, що особливо важливо у стрімко плинучому світі нових технології. Тому використання нейронної мережі у цій роботі вважається обґрунтованим.

Розглянемо, які існують релевантні нейронні моделі й методи для обраної у даній роботі предметної області кібербезпеки. Виявлення (розпізнавання) кібератак та вразливостей ІТС за допомогою нейронних мереж в цілому зводиться до оцінок величин параметрів безпеки ресурсів ІТС. Якщо виставлена за допомогою НМ оцінка перевищує певне граничне значення, то вважається, що кібератака виявлена. У протилежному випадку вважається, що рівень безпеки знаходиться в допустимих межах. У роботі [6] автори описали та проаналізовали різні нейромережеві методи та моделі:

- Методи простої та семантичної класифікації мережевих атак.
- Нейромережевий підхід виявлення SQL-ін'єкцій.
- Бінарний нейромережевий метод.
- Метод використання НМ гібридної структури типу CounterPropagation.
- Адаптивна система виявлення мережевих атак.
- Метод побудови сукупного класифікатора трафіку тощо.

Це є підтвердженням зацікавленості наукової спільноти у розвитку нейронних мереж у домені ІБ, але на сьогодні вони не є достатньо дослідженими.

Експертні системи та нейронні мережі тією чи іншою мірою успішно моделюють окремі сторони штучного інтелекту, використовуючи різні підходи. У той час як експертні системи використовують правила імплікації (IF-THEN), логічний висновок, нейронні мережі мають здатність гнучко навчатися і виконувати паралельну обробку даних.

Моделі машинного навчання змінюють парадигму експертних систем від тієї, в якій програміст повинен надавати правила та вхідні дані для отримання результатів (1.1), до такої, де вони можуть надавати вхідні дані та результати для виведення правил (1.2).

$$\text{вхідні дані} + \text{правила} \rightarrow \text{результати} \quad (1.1)$$

$$\text{вхідні дані} + \text{результати} \rightarrow \text{правила} \quad (1.2)$$

Програмне забезпечення зі стратегією (2) може претендувати на те, що отримані правила можуть бути застосовані до багатьох нових вхідних даних будь-ким, хто має програмне забезпечення, не вимагаючи, щоб вони самі володіли знаннями, необхідними для самостійного отримання результатів.

Ми бачимо, як це свідчить про нове покоління експертних систем, що базуються на нейронних мережах і глибокому навчанні. У даних ЕС експертиза є більш процедурною, ніж описовою, окрім цього нейронні експертні системи можуть оперувати з недостовірними та неповними даними.

Для даної конкретної задачі виявлення аномалій у запитах до веб-серверу є доречним застосувати нейронний модуль, замість класичного елемента ЕС – бази знань, на підставі зазначених переваг, а також для здатності подальшої моделі виявляти невідомі досі атаки нульового дня, бути здатною перенавчатись та загалом надавати власне

незалежне рішення для комплексної системи захисту та спеціалістів галузі. При цьому відзначимо необхідність наявності бази даних у запропонованому рішенні, щоб існувала можливість перенавчання НМ з врахуванням її помилок.

В даний час відомо багато вдалих прикладів застосування нейромережного підходу для побудови інтелектуальних інформаційних і, зокрема, експертних систем.

У дослідженні [7] автори пропонують обчислювальну систему на основі інтелектуальних гібридних моделей, яка за допомогою нечітких правил дозволяє будувати експертні системи в домені класифікації кібернетичних вторгнень, зосереджуючись на атаці SQL Injection.

Автори роботи [8] зазначають, що останнім часом для виявлення кіберзагроз, таких як мережева атака, проникнення шкідливого програмного забезпечення або фішинговий веб-сайт, використовувалося кілька моделей глибокого навчання. При цьому останні зазвичай страждають від того, що не можуть бути пояснені експертами з безпеки. Експертам з безпеки необхідно не тільки виявляти вхідні загрози, але й знати вбудовані функції, які спричиняють цей конкретний інцидент безпеки.

Тому Mahdavifar та інші [8] пропонують свою експертну систему глибокої вбудованої нейронної мережі (deep embedded neural network expert system, DeNNeS), яка отримує уточнені правила з архітектури навченої глибокої нейронної мережі (deep neural network, DNN) для заміни бази знань експертної системи. Пізніше база знань використовується для класифікації невидимого інциденту безпеки та інформування кінцевого користувача про відповідне правило, яке зробило цей висновок.

Таким чином, розглядаючи релевантні наукові роботи з цієї галузі, була доказана актуальність та життєздатність запропонованого підходу, тому можна формулювати вимоги до майбутньої нейронної експертної системи детектування шкідливого трафіку.

1.3 Постановка задачі та вимоги до експертної системи, що проектується

Враховуючи все вищезазначене, опишемо основні задачі та вимоги для цієї роботи. У роботі слід:

- Проаналізувати та визначити найнебезпечніші загрози для веб-ресурсів згідно рейтингів OWASP та CWE Top 10. На їх основі готова нейронна ЕС буде здатна детектувати та класифікувати конкретні кібератаки;
- Визначити підстави для організації нейронної експертної системи для детектування зловмисного трафіку. У роботі потрібно навести схематичне планування експертної системи на концептуальному рівні, продемонструвати зв'язок складових елементів зі застосуванням UML;
- Спроектувати та розробити базу даних для експертної системи та забезпечити захист її даних;

База даних передбачає створення резервних копій даних у двох випадках:

- Коли система дійсно виявила спробу атаки на веб-сервер у HTTP-запиті, потрібно зафіксувати поточний стан параметрів нейронної моделі;
- Коли система зробила помилку в розрахунках, у подальшому потрібно провести процедуру перенавчання, щоб не робити подібних помилок знову. Мета перенавчання – покращити здатність моделі виявляти усі види атак та якнайменше дедалі припускатися помилок при прогнозах. Але не завжди можна передбачити успішність пере-навчання моделі, тому все одно має бути можливість повернутися до попередньої моделі.

Таким чином, шляхом збереження основних параметрів нейронної моделі у БД, можна у будь-який момент повернутися до будь-якого колишнього стану нейронної моделі. Іншими словами, БД зможе описати еволюцію у навчанні та вдосконаленні моделі.

- Розробити модель нейронної мережі для експертної системи у якості бази знань. На вхід нейронній мережі надаються HTTP/URL запити. Як висновок вона повинна повернути рішення у вигляді ймовірності, наскільки даний запит є зловмисним, і, якщо так, визначити передбачений клас атаки. Обрана архітектура НС повинна бути обґрунтованою та проаналізованою у відповідності до найкращих показників ефективності;
- Поєднати розроблені елементи у єдину систему. Експертна система складається з нейронної мережі, бази даних та інтерфейсу для звичайного користувача (консольної програми-детектора). Вважається, що кінцевим користувачем системи

є спеціаліст з ІБ середнього рівню. Передбачається здатність системи коректно працювати як мінімум на двох операційних системах – Windows та Linux.;

- Протестувати та оцінити ефективність функціонування нейронної ЕС;
- Розробити рекомендації для застосування системи фахівцями з безпеки.

Підсумком роботи повинна стати розроблена нейронна ЕС, яка дозволяє:

- якнайшвидше детектувати кібератаки визначеного типу при аналізі трафіку до веб-серверу з точністю прогнозування не менше 90%;
- зберігати історію кібератак, вести облік, зберігати минулі стани нейронної мережі у базі даних;
- у разі помилок прогнозування експертної системи, вона має зберігати ці помилки для подальшого перенавчання.

Окремо можна виділити результат праці у виді розроблених рекомендацій зі застосування, документації для користувачів.

1.4 Визначення кібератак для подальшого детектування

Для того, щоб визначити головні загрози для веб-ресурсів, перш за все, виокремимо необхідність захисту конфіденційних (таємних, персональних, комерційних) даних, які зберігаються у базах даних, адже велика частка атак мають за мету здобуття ІзОД (паролів, даних облікових записів, фінансових рахунків, джерел отримання спеціального доступу та прав, наприклад, адміністратора).

По-друге, найбільш популярними є цільові атаки на сховища даних середніх комерційних організації – зокрема фінансових або банківських [5].

По-третє, визначимо найбільш ймовірні шляхи атак на веб-додатки згідно рейтингів OWASP та CWE Top 10.

OWASP (англ. Open Web Application Security Project) – це некомерційна організація, метою якої є підвищення обізнаності всіх фахівців галузі ІБ з питань розробки, експлуатації та захисту веб-додатків. OWASP Top 10 є популярним рейтингом з 10 найбільш небезпечних ризиків ІБ для веб-додатків, складений спільнотою експертів галузі.

OWASP Top 10 використовується в Microsoft, PCI DSS (Payment Card Industry Data Security Standard), MITRE (некомерційна організація, що займається розробками і дослідженнями в області системної інженерії за підтримки органів державної влади США) тощо. Попри це, рейтинг не покриває всі можливі уразливості і тому не є достатнім джерелом для повного захисту веб-додатків.

Рейтинг OWASP Top 10 за (2017-2021) роки представлений на рисунку 1.3, де можна побачити усі зміни критичності ризиків ІБ за минулі 4 роки [9].

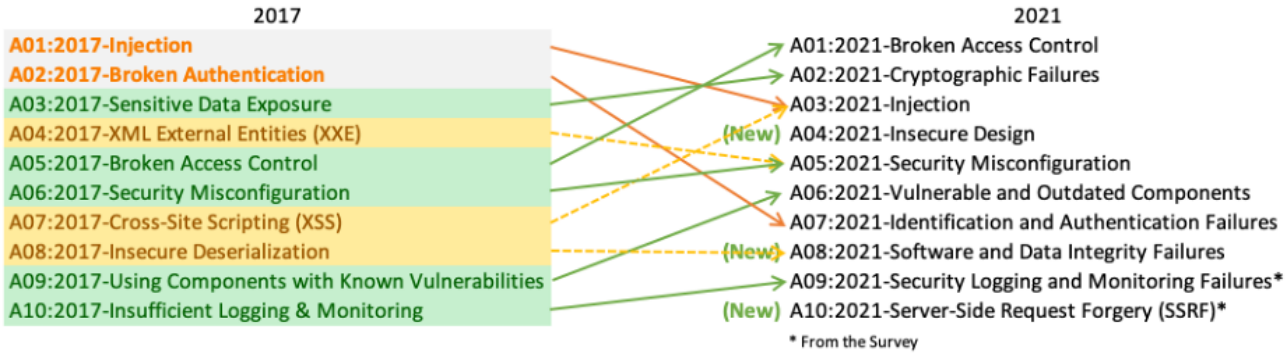


Рисунок 1.3 – Рейтинг OWASP Top 10 2021

CWE (англ. Common Weakness Enumeration) – загальний перелік вразливостей і недоліків безпеки програмного забезпечення (не обов'язково у веб-додатках), являє собою ієрархічний словник, призначений для розробників і фахівців щодо забезпечення безпеки ПЗ. CWE підтримується MITRE на замовлення Міністерства внутрішньої безпеки США і розвивається при широкій підтримці спільноти експертів.

Набір даних, проаналізований для розрахунку Топ-25 2022 року, містив загалом 37 899 записів CVE (англ. Common Vulnerabilities and Exposure) за попередні два календарні роки. Актуальний рейтинг десяти найбільш небезпечних загроз від CWE наведений у таблиці 1.1 [10].

Таблиця 1.1 – Рейтинг загроз CWE 2022.

№п\п	Позначення	Назва	Оцінка
1	CWE-787	Запис даних поза меж (Out-of-bounds)	64.20
2	CWE-79	Неправильна нейтралізація вводу під час генерації веб-сторінок (Міжсайтовий скріптинг, XSS)	45.97
3	CWE-89	Неправильна нейтралізація спеціальних елементів, що використовуються в команді SQL (Ін'єкція SQL)	22.11

Продовження таблиці 1.1

№п\п	Позначення	Назва	Оцінка
4	CWE-20	Неправильна перевірка вводу	20.63
5	CWE-125	Читання даних поза меж (Out-of-bounds)	17.67
6	CWE-78	Неправильна нейтралізація спеціальних елементів, що використовуються в команді ОС (Ін'єкція командного рядка ОС)	17.53
7	CWE-416	Використання вказівника кучі після її звільнення (Use-After-Free)	15.50
8	CWE-22	Неправильне обмеження шляху до каталогу з обмеженим доступом (Path Traversal)	14.08
9	CWE-352	Міжсайтова підробка запиту (CSRF)	11.53
10	CWE-434	Необмежене завантаження файлу небезпечного типу	9.56

Відзначимо, що нейронна мережа отримує на вхід та зможе розпізнавати шкідливий трафік, тобто вхідні дані (запити) до веб-серверів. Цей факт варто враховувати при виборі класів атак, тому що не всі атаки використовують вхідний запит при атаці. У даному випадку аналізу піддаються усі атаки типу ін'єкцій.

Отже, згідно рейтингів OWASP Top 10 та CWE Top 10, визначимо набір актуальних та додаткових атак, які спроектована експертна система має бути здатна розпізнавати [9]:

- SQL Injection. Атака SQL-ін'єкції полягає у вставці або «ін'єкції» SQL-запиту через вхідні дані від клієнта до програми. Успішний експлойт SQL-ін'єкції може зчитувати конфіденційні дані з бази даних, змінювати дані бази даних (Вставити/Оновити/Видалити), виконувати операції адміністрування бази даних (такі як завершення роботи СУБД), відновлювати вміст певного файлу, наявного у файлі СУБД. і в деяких випадках видають команди операційній системі.
- Cross Site Scripting – XSS. Тип ін'єкції, під час якого шкідливі сценарії (скріпти) впроваджуються на надійні веб-сайти. XSS-атаки відбуваються,

коли зловмисник використовує веб-програму для надсилання шкідливого коду, як правило, у формі сценарію сторони браузера, іншому кінцевому користувачеві.

- XML External Entities Injection (XEE). Атака на зовнішню сутність XML – це тип атаки на програму, яка аналізує вхідні дані XML. Ця атака відбувається, коли XML-вхід, що містить посилання на зовнішню сутність, обробляється слабко налаштованим аналізатором XML. Ця атака може призвести до розкриття конфіденційних даних, відмови в обслуговуванні, підробки запитів на стороні сервера, сканування портів з точки зору комп'ютера, на якому розташований парсер, та інших впливів на систему.
- Server-Side Includes Injection (SSI). SSI – це директиви, присутні у веб-додатках, які використовуються для передачі HTML-сторінки з динамічним вмістом. Атака на стороні сервера дозволяє використовувати веб-програму шляхом введення сценаріїв у сторінки HTML або віддаленого виконання довільних кодів.
- Buffer Overflow. Зловмисники використовують переповнення буфера, щоб пошкодити стек виконання веб-програми. Надсилаючи ретельно розроблені вхідні дані до веб-програми, зловмисник може змусити веб-програму виконати довільний код, фактично захопивши машину.
- Carriage Return Line Feed Injection (CRLFi). Термін CRLF стосується повернення каретки (ASCII 13, \r) Переведення рядка (ASCII 10, \n). Вони використовуються для позначення завершення рядка, однак у сучасних популярних операційних системах обробляються інакше. Атака CRLF Injection виникає, коли зловмиснику вдається надіслати CRLF у програму. Найчастіше це робиться шляхом зміни параметра HTTP або URL-адреси.
- XPath. Атаки XPath Injection виникають, коли веб-сайт використовує інформацію, надану користувачем, для створення запиту XPath для даних XML. Надсилаючи навмисно неправильну інформацію на веб-сайт, зловмисник може дізнатися, як структуровані XML-дані, або отримати доступ до даних, до яких він зазвичай не має доступу.

- Lightweight Directory Access Protocol (LDAP) – це програмний протокол, який використовується серверами для спілкування з локальними каталогами, і який зберігає та впорядковує дані для полегшення пошуку. Атаки LDAP схожі на атаки ін'єкції SQL. Ці атаки зловживають параметрами, які використовуються в запиті LDAP. У більшості випадків програма неправильно фільтрує параметри. Це може призвести до вразливого середовища, у яке хакер може ввести шкідливий код.
- Format String. Експлойт виникає, коли надіслані дані вхідного рядка оцінюються веб-додатком як команда. Format String є аргументом функції форматування та є рядком ASCII Z, який містить текст і параметри формату, наприклад: printf ("The magic number is: %d\n", 1911).

Як можна побачити, усі визначені класи кібератак за своїм сенсом близькі до ін'єкцій та використовують вхідні данні у формі модифікованого HTTP або URL-запиту, які посилають до веб-серверу. Цей факт робить можливим аналіз цих вхідних даних нейронною мережею однаковим чином.

1.5 Концептуальний та логічний опис системи

Уніфікована мова моделювання, або UML, є візуальною мовою, яка допомагає розробникам програмного забезпечення візуалізувати та створювати нові системи. В UML 2.0 існує дві основні категорії діаграм: структурні та поведінкові. Кожна діаграма UML належить до однієї з цих двох категорій діаграм. Призначення структурних діаграм — показати статичну структуру системи, що моделюється. Вони включають діаграми класів, компонентів і/або об'єктів. З іншого боку, діаграми поведінки показують динамічну поведінку між об'єктами в системі, включаючи їхні методи, співпраці та дії. Прикладами діаграм поведінки є діаграми діяльності, варіантів використання та послідовності.

Приклад діаграми класів для нейронної експертної системи наведений на рисунку 1.4. У ній описані головні сутності: Інтерфейс Користувача є головною сполучною ланкою між користувачем, класом Бази Даних і Нейронною Моделлю. Він координує роботу всієї системи відповідно до бажань користувача. Він ініціює

підключення до БД. Підключення та всі необхідні операції з модифікації БД виконує та реалізує клас База Даних. Клас Нейронної моделі є відповідальним за створення моделі, її навчання, тестування. Обробку великих текстових даних перед їх направленням до нейронної моделі здійснює Обробник Даних (DataProcessor), а саме препроцесинг, токенізацію, векторизацію. Клас UrlRequest є класом даних, він визначає сам запит, щоб у подальшому його простіше було зберігати у БД. У цій схемі передбачається, що на один Інтерфейс Користувача з безлічі можливих достатньо мати одну Нейронну Модель, конектор до БД і DataProcessor. Звісно, обробник даних може обробляти безліч UrlRequest.

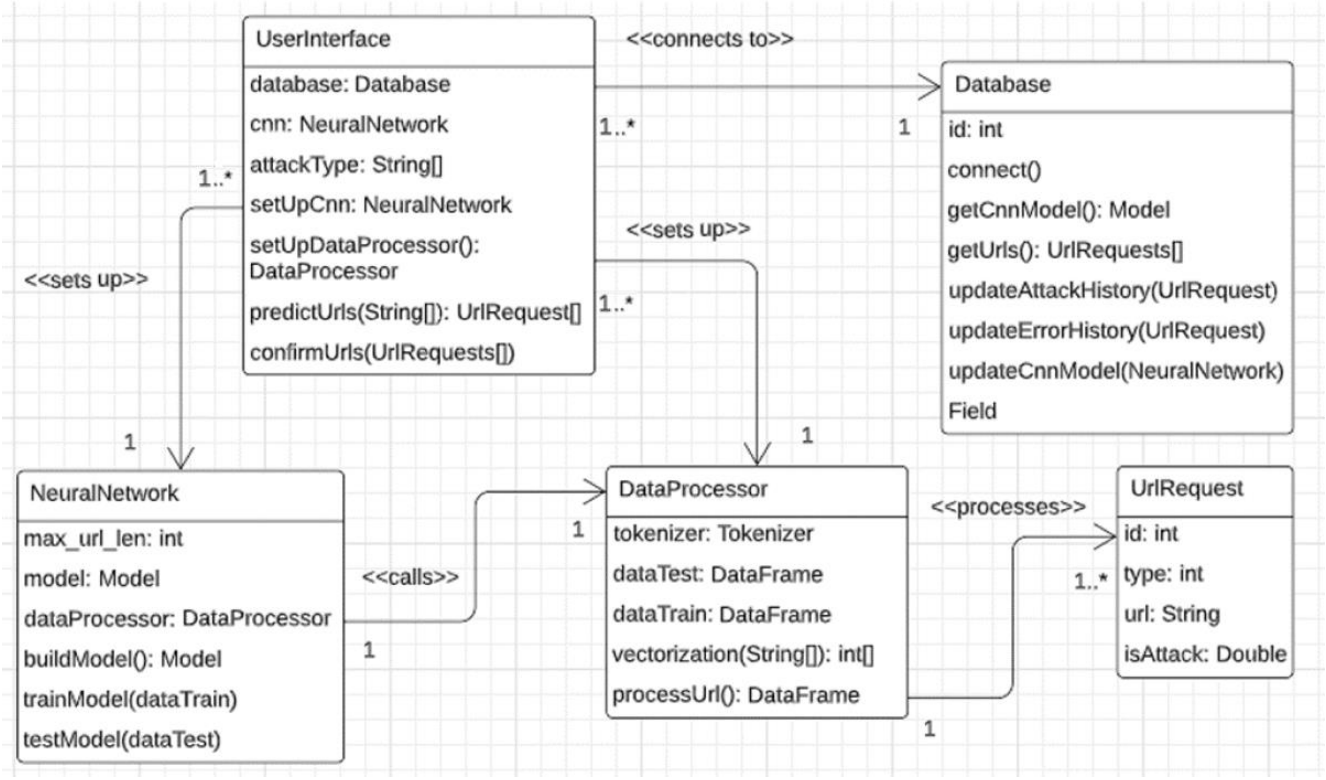


Рисунок 1.4 – Діаграма класів

Поведінка об’єкта є не лише прямим наслідком його вхідних даних, але також від його попереднього стану. Минулий стан об’єкта можна найкраще змоделювати за допомогою діаграми кінцевого автомата. UML State Machine Diagrams (або іноді їх називають діаграмою станів, кінцевим автоматом або діаграмою станів) показують різні стани системи. Діаграми кінцевого автомата також можуть показати, як об’єкт реагує на різні події, змінюючи один стан в інший. Діаграма кінцевого автомата — це діаграма UML, яка використовується для моделювання

динамічної природи системи. Діаграма кінцевого автомату для даної експертної системи представлена на рисунку 1.5.

Після запуску інтерфейсу користувача починається етап ініціалізації. Він включає різні етапи, які не помітні кінцевому користувачеві. Відбувається підкачування необхідних бібліотек, переважно для обробки великих даних та нейронної моделі, ініціалізуються змінні, також відбувається підключення до БД.

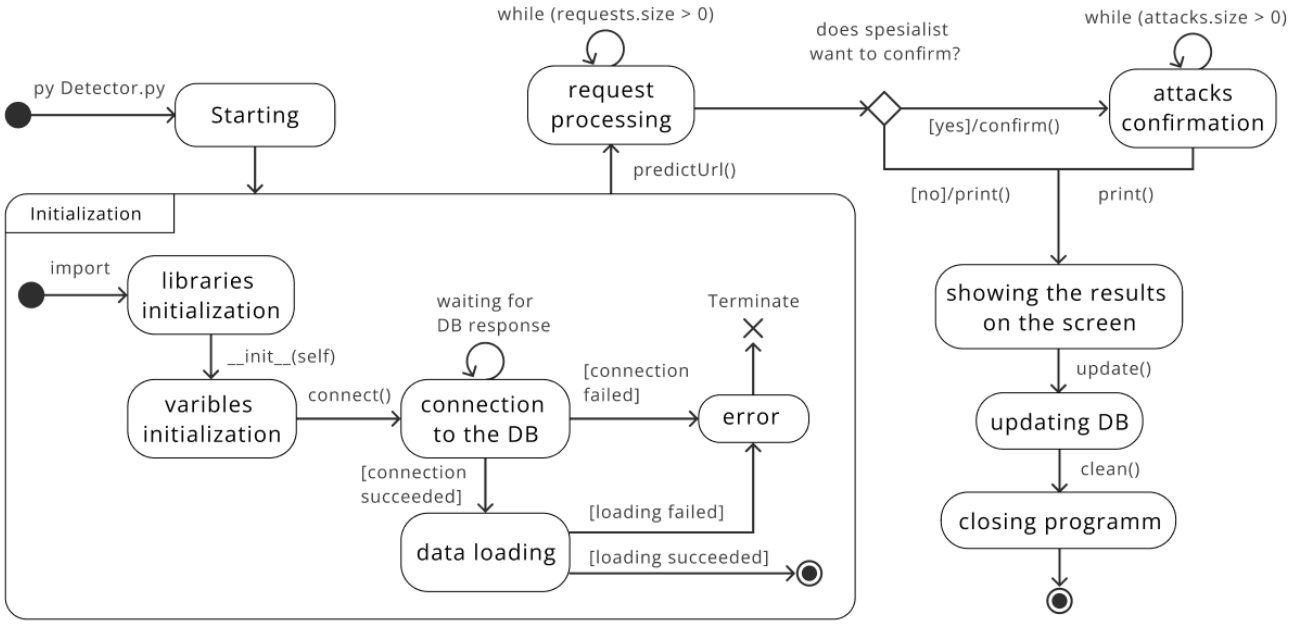


Рисунок 1.5 – Діаграма кінцевого автомату

Якщо підключення успішне, завантажується також остання, вже навчена, нейронна модель. Якщо підключення не відбулося, вся робота із системою надалі не можлива, процес зупиняється. Після ініціалізації нейронна модель здійснює обробку запитів, доки вони продовжують надходити. Передбачається, що результати передбачення показуються фахівцю з безпеки, який за бажання може їх виправити, якщо він не погоджується з рішенням моделі. У цьому разі відбувається виправлення результатів. Далі результати роботи моделі зберігаються у БД. Після цього програма завершує виконання.

Діаграми діяльності описують, як діяльність сутностей координується для надання послуги, яка може бути на різних рівнях абстракції. Як правило, подія має бути досягнута деякими операціями, особливо якщо операція призначена для досягнення кількох різних речей, які вимагають координації, або як події в одному

варіанті використання співвідносяться одна з одною, зокрема випадки використання, де дії можуть збігатися і вимагати узгодження.

Діаграма діяльності системи наведена на рисунку 1.6. Вона демонструє, як потік управління переходить від однієї діяльності до іншої, при цьому увага фіксується на результаті діяльності. Діаграма є логічним продовженням двох наведених вище схем, але з вищим рівнем деталізації.

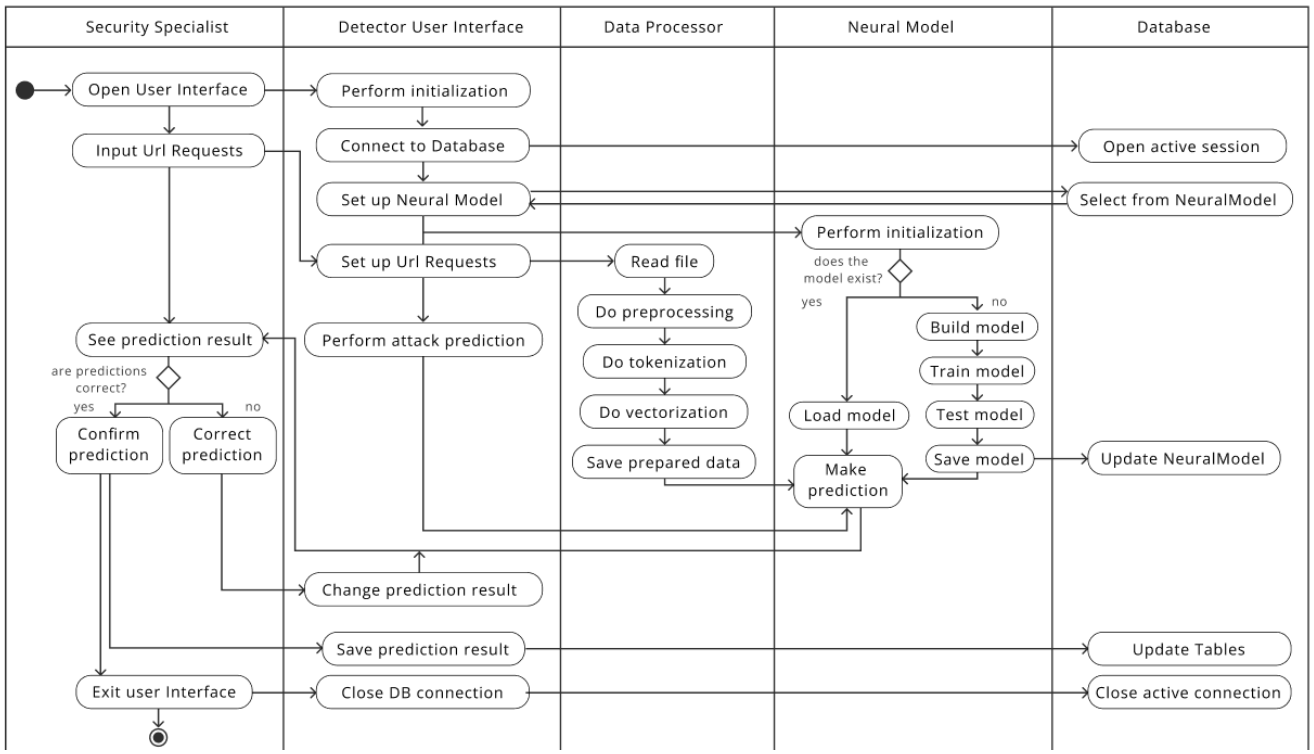


Рисунок 1.6 – Діаграма діяльності

Таким чином, UML дозволив описати систему з різних точок зору та розглянути різні аспекти поведінки системи, які не були помітні до проектування. У цьому випадку використання UML дозволило візуалізувати структуру експертної системи на рівні, зручному для подальшого програмування.

2 РОЗРОБКА БАЗИ ДАНИХ СИСТЕМИ

2.1 Обґрунтування необхідності створення бази даних

Дані – це найцінніший корпоративний актив для будь-якого існуючого підприємства та бізнесу. База даних (БД) – це спільний набір логічно пов'язаних даних і опис цих даних, призначений для задоволення інформаційних потреб зацікавлених користувачів. У даному випадку потреба полягає у збереженні інформації та можливості у будь-який момент повернутися до попереднього стану експертної системи [11].

Бази даних необхідно захищати та регулярно перевіряти актуальність цього захисту, адже робота усієї системи може залежати переважно від справної роботи БД. Використовуючи спеціальні методики, можна запобігти несанкціонованому доступу до бази даних, зміні, втраті або витоку конфіденційної або секретної інформації.

Захист баз даних – це комплексний підхід до забезпечення безпеки інформації, що зберігається в них. Під цим поняттям маються на увазі заходи, створені задля запобігання її втрати, розкрадання чи зміни [11].

Як зазначалося у попередньому розділі, існує необхідність розробити базу даних нейромережевого детектору зловмисного трафіку до веб-серверу з окремим упором на її власний захист, який є важливою складовою комплексного поняття захищеності усієї експертної системи.

Основною метою даного розділу є розробка відповідної бази даних та механізмів її захисту (зокрема, розмежування та контроль доступу за допомогою користувачів, ролей, профілів тощо).

З поставлених вимог відомо, що база даних передбачає створення резервних копій даних у двох випадках:

– Коли нейронна мережа дійсно виявила спробу атаки на веб-сервер у HTTP-запиті, потрібно зафіксувати поточний стан параметрів нейронної моделі.

– Коли нейронна мережа зробила помилку в розрахунках, у подальшому потрібно провести процедуру перенавчання, щоб не робити подібних помилок знову.

У описаній ситуації маємо наступні основні данні:

– НЕЙРОННА МОДЕЛЬ з атрибутами: model_id, model_file, model_date, model_accuracy, model_comment – це данні про нейронну мережу, її параметри, архітектуру, точність тощо.
– HTTP-ЗАПИТ з атрибутами: http_id, http_query, http_type, http_ date, http_danger_lvl – це дані про HTTP-запити веб-серверу.

Атрибут model_file зберігає у собі файл нейронної моделі з усіма параметрами нейронів, зі структурою моделі та зберігає вже навчений стан моделі у спеціальному форматі .h5. У нашому випадку, один такий файл може мати розмір приблизно 200-300 КБ, що не суттєво може впливати на загальний об'єм БД та швидкодію. Атрибут model_date відповідає за збереження дати, коли конкретна модель була завантажена у БД, тобто за цим атрибутом можна відстежити послідовність додавання нових моделей на часовій осі. Атрибут model_accuracy містить загальний показник точності моделі – це одна з метрик якості, яка називається accuracy. Атрибут model_comment існує для того, щоб можна було залишати якісь коментарі стосовно тієї чи іншої моделі.

Що стосується Запита, то у нього є такі характеристики як http_query, що є, власно, самим текстом (контентом) HTTP-запита. Атрибут http_type позначає спеціальний клас моделі, відповідний до атак, які здатна виявляти нейронна модель. Артибут http_ date позначає дату детектування запиту, вона необхідна, щоб відстежувати з'явлення нових запитів у часі. Атрибут http_danger_lvl – це імовірність загрози за розрахунками поточної моделі, можна сказати, що це передбачення моделі щодо запиту.

2.2 Проектування бази даних

Якість проектування бази даних може впливати на роботу з нею. З добре спроектованою базою даних легше працювати, легше писати запити.

Для якісного проектування бази даних існують різні методики, різні послідовності кроків чи етапів, які багато в чому схожі.

Загалом можна виділити такі основні етапи проектування [12]:

1) Виділення сутностей та їх атрибутів, які зберігатимуться у базі даних, і формування з них таблиць.
2) Визначення унікальних ідентифікаторів (первинних ключів) об'єктів, що зберігаються у рядках таблиці.
3) Визначення відносин між таблицями за допомогою зовнішніх ключів;
4) Нормалізація бази даних.

У цьому підрозділі застосуємо концептуальне проектування – це створення концептуального уявлення бази даних, що включає визначення типів найважливіших сутностей і існуючих між ними зв'язків і атрибутів, причому розроблена модель предметної області (ПрО) є незалежною від будь-яких фізичних аспектів її представлення [11].

У першому етапі відбувається виділення сутностей. Сутність (entity) є типом об'єктів, які повинні зберігатися в базі даних. Кожна таблиця у базі даних має представляти одну сутність. Кожна сутність визначає набір атрибутів. Атрибут є властивістю, яка описує деяку характеристику об'єкта. Кожен стовпець повинен зберігати один атрибут сутності. А кожен рядок представляє окремий об'єкт чи екземпляр сутності.

Виходячи з початкових даних, є типи сутностей: перша сутність – Нейронна Модель, друга сутність – HTTP-Запит.

Визначимо типи зв’язків між виявленими сутностями. Будь-яка існуюча, збережена Нейронна Модель може детектувати вразливості у будь-яких HTTP-Запитах. Можлива ситуація, коли Нейронна Модель не знайде ніяких HTTP-Запитів, які вона б визначила як небезпечні, наприклад, коли модель тільки почала свою роботу та ще не встигла знайти небезпечні запити.

ER-діаграма для сутностей Нейронна Модель, HTTP-Запит і зв'язків між ними більш наочно представлена на рис. 2.1.

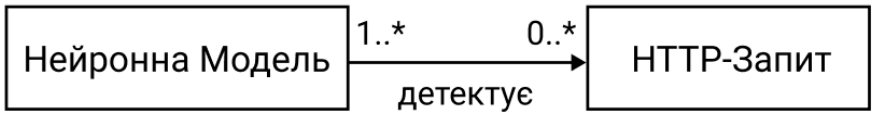


Рисунок 2.1 – ER-діаграма для сутностей Нейронна Модель, HTTP-Запит та зв’язків між ними

Далі йде етап визначення атрибутів і пов’язання їх з типами сутностей і зв’язків. Враховуємо, що при визначенні атрибутів відбувається поділ складних комплексних елементів більш прості. У результаті аналізу були виявлені атрибути та зв’язки з сутностями, як продемонстровано у Таблиця 2.1.

Таблиця 2.1 – Атрибути сутностей

Ім’я сутності	Атрибути
Нейронна Модель	model_id, model_file, model_date, model_accuracy, model_comment
HTTP-Запит	http_id, http_query, http_date, http_type, http_danger_lvl

У Таблиці 2.2 показаний фрагмент словника даних, в якому наведена інформація про атрибути сутностей Нейронна Модель і HTTP-Запит.

Далі відбувається визначення доменів атрибутів. Кожен атрибут має домен. Домен являє собою набір допустимих значень для одного або декількох атрибутів. Якесь інше значення, яке відповідає домену, атрибут мати не може.

Таблиця 2.2 – Фрагмент словника даних, яких містить опис атрибутів

Ім’я сутності	Атрибути	Опис	Тип і розмірність представлення даних	Null-значення	Багатозначний
Нейронна Модель	model_id	Однозначно визначає нейронну модель	Числовий	Ні	Ні
	model_file	Файл моделі	BLOB (Binary Large Object, Двійковий Великий Об'єкт)	Ні	Ні
	model_date	Дата додавання моделі до БД	Дата (дд.мм.гггг)	Ні	Ні
	model_accuracy	Точність передбачень моделі	Числовий	Ні	Ні
	model_comment	Коментар до моделі	Текст довжиною до 255 символів	Так	Ні
	http_id	Однозначно визначає HTTP-запит	Числовий	Ні	Ні

Продовження таблиці 2.2

HTTP-Запит	http_query	Текст запиту, контент	Текст	Ні	Ні
	http_date	Дата додавання запиту до БД	Дата (дд.мм.гггг)	Ні	Ні
	http_type	Клас запиту	Текст довжиною до 255 символів	Ні	Ні
	http_danger_lvl	Показник рівню небезпеки запиту, який визначила модель	Числовий	Ні	Ні

Окрім цього, необхідно проаналізувати, а чи може атрибут мати відсутнє значення, тобто значення NULL, та чи може атрибут бути багатозначним. У таблиці 2.3 є фрагмент словника даних, який описує домени для кожного з визначених атрибутів кожної сутності.

Таблиця 2.3 – Фрагмент словника, який містить опис доменів атрибутів

Ім’я атрибуту	Характеристика домена	Пояснення і обмеження для домену	Приклади допустимих значень
model_id	Автоматична вставка послідовних чисел при додаванні моделі	Числовий автоінкремент, неможливо змінити вручну	Натуральні числа, наприклад, 3
model_file	BLOB (Binary Large Object, Двійковий Великий Об'єкт)	Файл формату .h5, розміром не більше 1 гігабайта	Завантажений файл вказаного формату
model_date	Дата	Короткий формат дати (дд.мм.рррр)	25.03.2014
model_accuracy	Числові значення	Дійсне число > 50.0	88,5
model_comment	Короткий текст	Текст довжиною до 255 символів	Model stable to SQLi,…
http_id	Автоматична вставка послідовних чисел при додаванні запита	Числовий автоінкремент, неможливо змінити вручну	Натуральні числа, напр., 1
http_query	Довгий текст	Текст за розміром не більше 1 гігабайта, відображаються тільки перші 64000 символи	GET /tienda1/publ
http_date	Дата	Короткий формат дати (дд.мм.рррр)	25.03.2014
http_type	Короткий текст	Текст довжиною до 255 символів	SQLi
http_danger_lvl	Числові значення	Дійсне число > 50.0	54,5
model_date	Дата	Короткий формат дати (дд.мм.рррр)	25.03.2014

Метою наступного етапу є визначення всіх потенційних ключів для кожного типу сутності і, якщо таких ключів виявиться кілька, вибір серед них первинного ключа.

Потенційний ключ – це такий атрибут або безліч атрибутів, які єдино ідентифікують кортеж у відношенні (властивість унікальності), і при цьому не можна виділити підмножину атрибутів із знайдених, яка буде також дотримуватися властивості унікальності (властивість мінімальності) [12].

Сутність «Нейронна Модель» – потенційні ключі: model_id, model_file. Сутність «HTTP-Запит» – потенційні ключі: http_id, http_query.

Первинний ключ – такий єдиний потенційний ключ, який був обраний для ідентифікації кортежів у відношенні [12].

– Сутність «Нейронна Модель» – первинний ключ model_id.
– Сутність «HTTP-Запит» – первинний ключ http_id.

Основні етапи побудови концептуальної моделі завершені, отже можна проводити розробку логічної схеми БД.

Логічне проектування – процес створення логічної схеми бази даних на основі обраної моделі організації даних – у роботі використовується реляційна модель; перетворення концептуального уявлення в логічну структуру бази даних, включаючи проектування відношень.

Перш за все, потрібно провести виняток особливостей, які несумісні з реляційною моделлю. При цьому етапі уточняється концептуальна модель з метою усунення особливостей, несумісних з реляційною моделлю.

Призначення даного етапу полягає в наступному: видалення двосторонніх зв'язків "багато-до-багатьох"(*:*); видалення рекурсивних зв'язків "багато-до-багатьох" (*:*); видалення складних зв'язків; видалення багатозначних атрибутів. У концептуальній моделі є зв'язок "багато до багатьох" (*:*), тому треба виконати декомпозицію цього зв'язку для виявлення проміжної сутності. При цьому, спираючись на можливість моделі виявляти не тільки запит-атаку, а ще й робити помилку у виявленні запита, доцільно заздалегідь поділити усі запити на дві категорії: запити-помилки та запити-атаки.

Отже, у сутності HTTP-Запит з'являється новий атрибут-ознака наявності помилки: помилка_атаки або attack_error. Якщо значення нового атрибуту – 1, то Нейронна Модель здійснила помилку в розрахунках, якщо значення дорівнює 0, то помилки немає, та атака дійсно існує.

Попередній зв'язок замінюється двома зв'язками "один до багатьох" (1:*), в яких бере участь нова сутність-посередник, що показано на рис. 2.2 – «Помилка» (Моделі) та на рис. 2.3 – «Атака» (яку виявила Модель).

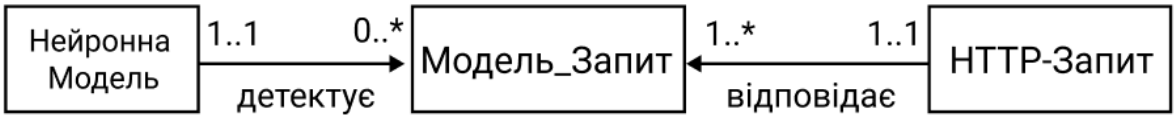


Рисунок 2.2 – Декомпозиція зв'язку «Нейронна Модель – детектує – HTTP-Запит»

Маємо фінальний зв'язок, який представлений на рисунку 2.3.

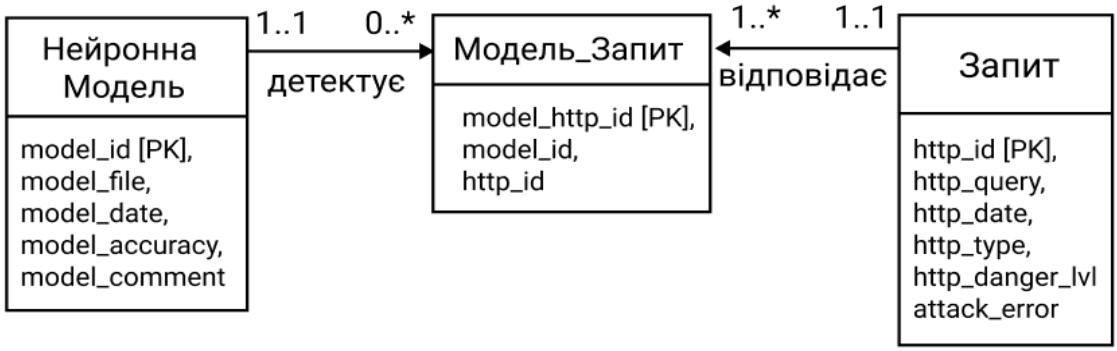


Рисунок 2.3 – ER-діаграма логічної схеми БД

Поточна модель не містить складних зв'язків, багатозначних атрибутів і рекурсивних зв'язків *:*.

Наступним кроком є перевірка відношень логічної схеми БД з використанням методів нормалізації. Схема відношень вже знаходиться у першій нормальній формі (1НФ), адже: кожен перетин рядка і стовпця містить рівно одне (атомарне) значення з відповідного домену.

Змінна відношення знаходиться у 2НФ, якщо вона знаходиться у 1НФ і кожен її атрибут, який не входить до складу первинного ключа, характеризується повною функціональною залежністю від цього первинного ключа.

Розглянемо сутність HTTP-Запит з ПК – http_id. Можна розглянути атрибути та побачити, що не існує функціональної залежності між ПК та атрибутом http_type.

Клас запиту – це його приналежність до заздалегідь визначених Класів атак, сам HTTP-Запит не може визначати свій Клас.

Отже, виділяємо нову сутність КласЗапиту та визначаємо залежність *:1 між ними: множина HTTP-Запитів описується одним із Класів (див. рис. 2.4). Інші відношення вже знаходяться у 2НФ.

Рисунок 2.4 – Перетворення відношення «Запит» з 1НФ до 2НФ

Відношення знаходиться в третій нормальній формі (3НФ), якщо воно знаходиться в 1НФ і в 2НФ і не має атрибутів, що не входять до складу первинного ключа, які перебували б у транзитній функціональній залежності від цього первинного ключа. Згідно з цим визначенням, схема знаходиться у 3НФ (рис. 2.5).

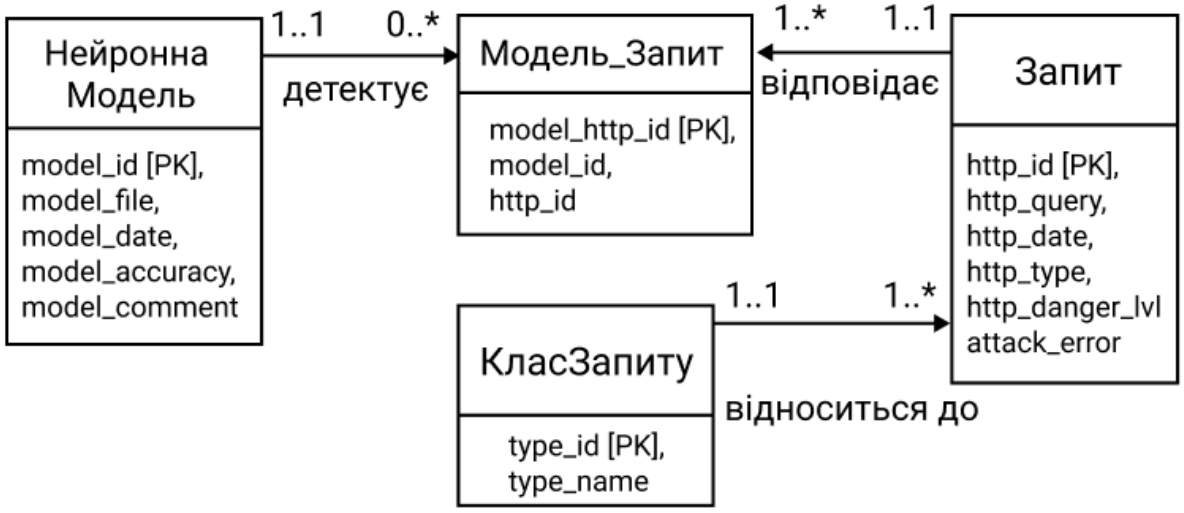


Рисунок 2.5 – ER-діаграма логічної схеми БД

Для розробки використаємо середу Oracle Database – це об'єктно-реляційна система управління базами даних (СУБД) компанії Oracle. Для виконання дій над конкретною БД скористуємось інструментом SQL*Plus.

SQL*Plus — це програма-інтерпретатор командного рядка для роботи з системою керування базами даних Oracle Database, в якій можуть виконуватися команди SQL і PL/SQL в інтерактивному вигляді або зі сценарію. SQL*Plus працює як відносно простий інструмент з інтерфейсом командного рядка. Програмісти та адміністратори СУБД зазвичай використовують його як інструмент за замовчуванням, тому що інтерфейс доступний практично у будь-якій установці програмного забезпечення Oracle.

На рис. 2.6 продемонстрований процес створення користувача БД від імені адміністратора БД, надання усіх привілеїв новому користувачу та підключення до сеансу з БД користувача.

```
SQL> create user student identified by student;

User created.

SQL> grant all privileges to student;

Grant succeeded.

SQL> connect student/student
Connected.
```

Рисунок 2.6 – Створення користувача БД

Від імені student необхідно створити усі задані на поточній ER-моделі таблиці. На рис. 2.7 показаний SQL запит, який створює таблицю для сутності КласЗапиту (HTTP_TYPE).

```
SQL> create table HTTP_TYPE
  2  (
  3  type_id integer primary key not null,
  4  type_name varchar(30) not null,
  5  unique (type_id, type_name)
  6  );

Table created.
```

Рисунок 2.7 – Створення таблиці БД HTTP_TYPE

На рис. 2.8 показаний SQL запит, який створює таблицю HTTP-Запитів (HTTP_Request). При цьому, слід звернути увагу, що у процесі створення таблиці був заданий FOREIGN KEY http_type_id, який посилається на таблицю HTTP_TYPE, що вимагає від значень http_type_id, щоб вони посилались на значення type_id.

```
SQL> create table HTTP_Request (
  2  http_id integer primary key not null,
  3  http_query varchar2(300) not null unique,
  4  http_date date not null,
  5  http_type_id integer not null,
  6  http_danger_lvl float not null,
  7  FOREIGN KEY (http type id) REFERENCES student.HTTP TYPE(type id)
  8  );

Table created.
```

Рисунок 2.8 – Створення таблиці БД HTTP_Request

На рис. 2.9 показаний SQL запит, який створює таблицю Нейронних Моделей (NeuroModel). Її значення не залежать від значень інших таблиць, проте, як і усі інші, значення мають бути не рівними NULL, окрім коментаря (model_comment), який є опціональним за своїм визначенням.

```
SQL> create table NeuroModel (
  2  model_id integer primary key not null,
  3  model_file blob not null,
  4  model_date date not null,
  5  model_accuracy float not null,
  6  model_comment varchar2(300)
  7  );

Table created.
```

Рисунок 2.9 – Створення таблиці БД NeuroModel

На рис. 2.10 показаний SQL запит, який створює таблицю Модель_Запит (Model_Request).

```
SQL> create table Model_Request (
  2  model_http_id integer primary key not null,
  3  model_id integer not null,
  4  http_id integer not null,
  5  foreign key (model_id) references student.NeuroModel(model_id),
  6  foreign key (http_id) references student.HTTP_Request(http_id)
  7  );

Table created.
```

Рисунок 2.10 – Створення таблиці БД Model_Request

У останній таблиці, згідно ER-діаграми, були передбаченні два посилання (foreign key) на значення з таблиць NeuroModel та HTTP_Request.

На рис. 2.11 зображені дані таблиці HTTP_Request.

```
SQL> select * from HTTP_Request;

   HTTP_ID HTTP_QUERY                    HTTP_DATE HTTP_TYPE_ID HTTP_DANGER_LVL
---------- ----------------------------- --------- ------------ ---------------
         1 GET /tienda1/publico/caracteri 12-MAY-21            1             .93
           sticas.jsp?id=d%27z%220

         2 POST /tienda1/publico/autentic 12-MAY-21            1             .88
           ar.jsp?login=61%27%20OR%20%276
           1%27=%2761&pwd=FrAmE30.&rememb
           er=&modo=entrar
```

Рисунок 2.11 – Дані у таблиці HTTP_Request

Обмежимося існуванням наступних класів моделі: «SQLi», «XSS», «SSI», «BufferOverflow», «CRLFi», «XPath», «LDAPi», «FormatString», також у разі неможливості вирішення конкретного типу атаки (або наявності атаки нульового дня) існує загальне рішення – «anomalous», та у разі незловмисного запиту – «normal» (див. рис. 2.12).

```
SQL> select * from HTTP_TYPE
  2  ;

   TYPE_ID TYPE_NAME
---------- ----------------------
         1 SQLi
         2 anomalous
         3 normal
         4 XSS
         5 SSI
         6 BufferOverflow
         7 CRLFi
         8 XPath
         9 LDAPi
        10 FormatString

10 rows selected.
```

Рисунок 2.12 – Дані у таблиці HTTP_TYPE

На рис. 2.13 зображені дані таблиці NeuroModel. У даному випадку є тільки одна модель нейронної мережи. Атрибут model_file при цьому містить hex-код файлу моделі. Ця модель відібрала 2 запити, які тепер збережені у таблиці HTTP_Request.

```
SQL> select * from Model;

  MODEL_ID MODEL_FILE           MODEL_DAT MODEL_ACCURACY MODEL_COMMENT
---------- -------------------- --------- -------------- ---------------
         1 894844460D0A1A0A0000 03-MAY-21           95.2 first try
           0000000808004001000
           0000000000000000000
           0000FFFFFFFFFFFFFFF
           7073050000000000FFFF
           FFFFFFFFFFFF00000000
           0000000060000000000
           0000010000000000000
```

Рисунок 2.13 – Дані у таблиці Model

Зв'язок між даними NeuroModel та HTTP_Request забезпечує таблиця Model_Request (див. рис. 2.14).

```
SQL> select * from Model_Request;

MODEL_HTTP_ID   MODEL_ID    HTTP_ID
------------- ---------- ----------
            1          1          1
```

Рисунок 2.14 – Дані у таблиці Error

2.3 Розробка механізмів захисту бази даних

Створеною базою даних будуть користуватися насамперед фахівці з інформаційної безпеки (ІБ). Враховуючи те, що в галузі безпеки доцільно враховувати різні думки різних експертів, щоб дійти чогось спільно-го, доступ до бази даних має бути наданий широкій аудиторії.

Припустимо, що ми маємо фахівців різного рівня, яким будуть надані різні можливості (привілеї доступу), відповідно до їх рівня кваліфікації. У даному випадку є молодший співробітник – junior, є більш досвідчений – senior.

Усі користувачі мають безпосередньо підключатися до БД (створювати сесію), обирати значення рядків або дивитися (select) усі таблиці. Адміністратор має право видаляти, додавати та оновляти значення таблиці.

Також потрібно реалізувати політики обмеження ресурсів БД між користувачами. Необхідно обмежити кількість сеансів з БД до 1, адміністратор БД (АБД) та користувач, який створив таблиці, може мати 2 сесії. Також необхідно обмежити кількість невдалих спроб входу до системи від імені будь-якого користувача п'ятьма, при цьому наступна спроба буде доступна тільки за добу. Паролі усіх користувачів повинні бути змінені раз у місяць.

Для підключення до бази даних користувач повинен мати обліковий запис. Слід розрізняти поняття користувача та обліковий запис. Користувач – людина, яка має обліковий запис, за допомогою якої визначаються його повноваження та дозволи в межах бази даних. Для керування обліковими записами використовуються такі конструкції мови SQL: CREATE USER – створення облікового запису; ALTER USER – для зміни наявного облікового запису; DROP USER – видалення облікового запису [12].

У даному випадку, припустимо, що є 4 спеціалісті з інформаційної безпеки: security1, security2, security3, security4. Процес їх створення продемонстрований на рис. 2.15. При цьому, вже створеними та привілейованими є користувачі student (має усі права, створював усі таблиці) та sys (адміністратор).

```
SQL> create user security1 identified by q1w2e3r4t5;

User created.
SQL> create user security2 identified by qwert12345;

User created.

SQL> create user security3 identified by qwertyuiop;

User created.

SQL> create user security4 identified by 1234567890;

User created.
```

Рисунок 2.15 – Створення користувачів БД

За замовчуванням в Oracle користувач, після його створення нічого не може зробити, навіть підключиться до БД. Для того щоб він міг підключатися до бази даних, працювати з нею, з об'єктами бази даних та даними йому необхідно видати дозволи на ті чи інші дії.

Для користувачів існують такі види привілеїв [13]:

– Об'єктні (Object privileges) – це дозволи на об'єкти схеми, такі як таблиці, уявлення, послідовності, пакети тощо. Для використання об'єктів схеми, що належать іншому користувачеві, потрібні привілеї на цей об'єкт.
– Системні (System privileges) – це дозволи на операції рівня бази да-них, наприклад, підключення до бази даних, створення користувачів, внесення змін до конфігурації бази даних тощо.

Ресурси бази даних (час центрального процесора, паралельно виконуються сеанси, операції логічного читання, час підключення тощо) не є необмеженими, отже, АБД повинен керувати та розподіляти їх між користувачами бази даних.

Oracle надає механізм контролю обсягу ресурсів, який може витрачати користувач. Таким механізмом є профіль (PROFILE) - іменований набір обмежень ресурсів, який може бути наданий користувачеві (вказує, які системні ресурси може задіяти цей користувач).

Наприклад, за допомогою профілю можна обмежити використання користувачем процесорного часу або числа операцій введення-виводу. Для створення профілю використовується команда CREATE PROFILE, в якій вказуються обмеження, які потрібно увімкнути. Не вказані обмеження будуть взяті від поточних значень профілю DEFAULT.

Профілі можуть бути також використані як механізм керування па-ролями: для контролю створення, повторного використання та підтвердження паролів користувачів, які застосовуються під час автентифікації.

На рис. 2.16 представлена команда створення профілю common_limit.

```
SQL> create profile common_limit limit
  2  password_life_time 30
  3  password_grace_time 7
  4  failed_login_attempts 5
  5  password_lock_time 1
  6  connect_time 60
  7  sessions_per_user 1;

Profile created.
```

Рисунок 2.16 – Створення профілю у БД

Реалізовувались наступні обмеження [14]:

- PASSWORD_LIFE_TIME 30 – кількість діб можливого використання пароля, перш ніж він має бути зміненим дорівнює 30. Якщо пароль не буде змінено протягом PASSWORD_GRACE_TIME, він має бути змінений, перш ніж можна буде повторити спробу входу до системи.
- PASSWORD_GRACE_TIME 7 – кількість діб, після яких пароль, що пройшов, повинен бути змінений дорівнює 7. Якщо його не буде змінено протягом цього періоду, обліковий запис вважається закінченим і пароль повинен бути змінений, перш ніж користувач зможе успішно увійти до системи.
- FAILED_LOGIN_ATTEMPTS 5 – кількість невдалих спроб входу в систему, перш ніж обліковий запис буде заблоковано дорівнює 5.
- PASSWORD_LOCK_TIME 1 – Число діб, протягом яких буде за-блоковано обліковий запис після перевищення спроб FAILED_LOGIN_ATTEMPTS, дорівнює 1. Після цього проміжку часу обліковий запис буде автоматично розблоковано.

– CONNECT_TIME 60 – максимальний загальний витрачений час на підключення у хвилинах дорівнює 60.
– SESSIONS_PER_USER 1 – максимальна кількість сеансів, одночасно дозволених користувачеві дорівнює 1.

Будь-який профіль не буде працювати та мати сенсу, якщо його не назначити хоча б одному користувачу. На рис. 2.17 назначимо загальний профіль усім користувачам, окрім student та sys.

```
SQL> alter user security1 profile common_limit;

User altered.

SQL> alter user security2 profile common_limit;

User altered.

SQL> alter user security3 profile common_limit;

User altered.

SQL> alter user security4 profile common_limit;

User altered.
```

Рисунок 2.17 – Присвоєння користувачам профілю

Управління безпекою через призначення прав безпосередньо користувачам має місце, але має два недоліки. По-перше, це може бути витратним завданням: додатки з тисячами таблиць і користувачів вимагають мільйонів дозволів. По-друге, якщо права були видані користувачеві, вони працюють завжди незалежно від обставин.

Обидві проблеми вирішуються використанням ролей (roles). Роль – це набір системних та об'єктних прав, які можуть бути видані та відкликані як єдине ціле, і після додавання цієї ролі, можуть бути тимчасово активовані та деактивовані під час існування сесії. Ролі можна створювати, змінювати та видаляти.

На рисунках 2.18, 2.19 були створені дві ролі: перша роль призначена для молодшого співробітника (junior), у якого менше привілеїв, а друга - для старшого (senior), який може, наприклад, видаляти користувачів та створювати нові ролі.

Ролі junior були надані такі системні права:

– create session - дозволяє створювати сеанс з базою даних;
– create any table - дозволяє створювати будь-яку таблицю у будь-якій схемі об'єктів;

– create any view - дозволяє створювати уявлення у будь-якій схемі об'єктів;
– select any table - дозволяє виконувати запити до таблиць, уявлень будь-якої схеми об'єктів.

Ролі senior були надані такі ж самі системні права, а також наступні:

– alter user - дозволяє змінювати параметри для будь-якого користувача: пароль іншого користувача, розпізнавальний метод, призначати квоти на будь-яку табличну область, встановлювати значення за замовчуванням та тимчасову табличну область, призначати профіль та задані за умовчанням ролі;
– create role - дозволяє створювати ролі;
– drop user - дозволяє видаляти користувачів.

```
SQL> create role junior;
Role created.
SQL> grant create session to junior;
Grant succeeded.
SQL> grant create any table to junior;
Grant succeeded.
SQL> grant create any view to junior;
Grant succeeded.
SQL> grant select any table to junior;
Grant succeeded.
```

Рисунок 2.18 – Створення ролі junior у БД

```
SQL> create role senior;
Role created.
SQL> grant create session to senior;
Grant succeeded.
SQL> grant create any table to senior;
Grant succeeded.
SQL> grant create any view to senior;
Grant succeeded.
SQL> grant select any table,alter user,create role,drop user to senior;
```

Рисунок 2.19 – Створення ролі senior у БД

Використовуємо створені ролі, щоб одразу надати користувачам певний набір привілеїв (рис. 2.20). Згодом зміни в ролях автоматично змінюватимуть права у всіх користувачів, яким була дана ця роль.

```
SQL> grant junior to security1, security2, security3, security4;

Grant succeeded.

SQL> conn /as sysdba
Connected.
SQL> grant senior to student;

Grant succeeded.
```

Рисунок 2.20 – Створення ролі senior у БД

Перевірка авторизації користувачів (див. рис. 2.21) показала, що усі створені користувачі можуть увійти до системи при заданих ідентифікаторах.

```
SQL> conn security1/q1w2e3r4t5;
Connected.
SQL> conn security2/qwert12345;
Connected.
SQL> conn security3/qwertyuiop;
Connected.
SQL> conn security4/1234567890;
Connected.
SQL> conn student/student;
Connected.
```

Рисунок 2.21 – Авторизація користувачів у систему

Система гарантує, що при іншому ідентифікаторі, доступ наданий не буде (див. рис. 2.22). Достатньо перевірити два облікових записи.

```
SQL> conn security1/q1w2e3;
ERROR:
ORA-01017: invalid username/password; logon denied


Warning: You are no longer connected to ORACLE.
SQL> conn security2/qwe33333;
ERROR:
ORA-01017: invalid username/password; logon denied
```

Рисунок 2.22 – Авторизація користувачів у систему

Перевірка виконання назначеного профілю common_limit показала, що застосовані обмеження справно працюють. Достатньо перевірити дію на одному обліковому записі. На рис. 2.23 перевірена неможливість входу у систему, якщо вже є одна поточка сесія з користувачем.

```
Enter user-name: security2
Enter password:
ERROR:
ORA-02391: exceeded simultaneous SESSIONS_PER_USER limit


Enter user-name:
```

SQL Plus

```
SQL> conn security2/qwert12345;
Connected.
SQL>
```

Рисунок 2.23 – Перевірка профілю common_limit

На рисунці 2.24 перевірене блокування облікового запису після 5-ти невдалих спроб входу.

```
SQL> conn security1/q1w2;
ERROR:
ORA-01017: invalid username/password; logon denied


SQL> conn security1/q1w2;
ERROR:
ORA-01017: invalid username/password; logon denied


SQL> conn security1/q1w2;
ERROR:
ORA-28000: The account is locked.
```

Рисунок 2.24 – Перевірка профілю common_limit

Перевірка ролі junior (див. рис. 2.25) показала, що застосовані обмеження справно працюють. Користувач може створювати сесію з системою, читати таблиці, але не може видаляти дані з них. Достатньо перевірити дію на одному обліковому записі.

```
SQL> conn security1/q1w2e3r4t5;
Connected
SQL> select * from student.Model_Request;

MODEL_HTTP_ID   MODEL_ID    HTTP_ID
------------- ---------- ----------
            1          1          1

SQL> delete student.Model_Request where  model_http_id = 1;
delete student.Model_Request where  model_http_id = 1
               *
ERROR at line 1:
ORA-01031: insufficient privileges
```

Рисунок 2.25 – Перевірка ролі junior

Перевірка ролі senior (див. рис. 2.26) показала, що застосовані обмеження справно працюють. Користувач student може створювати сесію з системою, читати таблиці, може видаляти дані з них та додавати.

```
SQL> conn student/student;
Connected.
SQL> select * from student.Model_Request;

MODEL_HTTP_ID   MODEL_ID    HTTP_ID
------------- ---------- ----------
            1          1          1

SQL> delete student.Model_Request where model_http_id = 1;

1 row deleted.

SQL> insert into student.Model_Request
  2  values (1,1,1);

1 row created.
```

Рисунок 2.26 – Перевірка ролі senior

Забезпечений захист та спроектована БД відповідають необхідному рівню безпеки, розроблене рішення відповідає поставленим вимогам предметної області, тому поставлені задачі вважається досягнутою.

3 РОЗРОБКА НЕЙРОННОЇ ЕКСПЕРТНОЇ СИСТЕМИ

3.1 Розробка моделі нейронної мережі

Метою даного розділу є розробка експертної системи, яка складається з нейронної мережі, бази даних та інтерфейсу користувача. Оскільки в минулому розділі вже було спроектовано та реалізовано базу даних, наступним кроком буде розробка архітектури нейронної мережі та її програмна реалізація.

Архітектура багатошарових нейронних моделей передбачає наявність різних шарів згрупованих нейронів. Окремий шар приймає на вхід вхідні сигнали, які є однаковими для кожного нейрона з групи. Число нейронів в шарі є незалежним параметром, який може обиратися та налаштовуватися найбільш вдалим чином відповідно до задачі. При цьому, вхідний вектор сигналів надходить до найпершого вхідного шару мережі, який завжди присутній, а вихідний вектор результатів є виходом останнього шару мережі. У цьому сенсі, можна сказати, що архітектура НМ володіє властивістю послідовності, тому що вихід кожного шару є входом наступного шару моделі.

Вся сутність функціонування мережі зводиться до існування певного правила для кожного нейрону, за яким значення вхідних параметрів перетворюються на значення вихідних параметрів шару. Це правило носить назву функції активації. Безпосередньо вихідне значення є ступенем активності нейрона. У ролі функцій активації можуть виступати абсолютно будь-які математичні функції відповідно до задачі, однак наступні пройшли перевірку часом та є найбільш відомими [15]:

– Лінійна функція активації створюватиме лише додатні числа у всьому діапазоні дійсних чисел.
– Сигмоїдна функція активації створюватиме лише додатні числа від 0 до 1. Сигмоїда є найбільш корисною та популярною для навчання даних, які також знаходяться між 0 та 1.

Навчання нейронної мережі – це процес, в якому параметри нейронної мережі модифікуються за допомогою послідовного надання їй елементів тестового набору

даних, де кожний є близьким до майбутнього реального екземпляру даних. Існує декілька типів навчання, що відповідають підходам до коригування параметрів.

Архітектура моделі обирається на базі проблеми, яку потрібно вирішити. Довге практичне застосування мереж дозволило вилучити декілька загальних напрямів задач, для яких доцільно використати НМ: розпізнавання образів і класифікація, прийняття рішення та керування процесами, кластеризація, прогнозування, оптимізація, апроксимація, стиснення даних і асоціативна пам'ять.

Для реалізації НМ виберемо відповідну архітектуру моделі для задачі прогнозування і класифікації кібератак. Спочатку потрібно виявити, чи може вхідний запит заподіяти шкоду або викрити конфіденційну інформацію, для цього НМ повинна відповідати на питання, чи містить запит ін’єкцію, що є бінарною відповіддю: «зловмисний запит» та «звичайний запит».

Проте, у даному випадку буде проводитись класифікація з декількома класами, адже, як показав аналіз рейтингів, ін'єкційні атаки можуть мати різний тип: SQL-ін’єкції, впровадження XML, JSON або JS-коду в запит, БД, HTML-код сторінки. Окрім цього, також виділимо клас anomalous, який відображає атаку «нульового дня». Опис конкретних класів буде визначатися згідно того набору даних, який буде обраний для навчання моделі. Існує багато моделей, які здатні провести подібну класифікацію: K-NN, класифікатор Naive Bayes, підтримка векторних машин (SVM), НМ та моделі без нагляду (вчителя).

Для вирішення даної задачі класифікації пропонується дослідити згорткову нейронну мережу, K-NN, класифікатор Naive Bayes та Decision Tree та обрати найбільш точну та ефективну.

Згорткові нейронні мережі (ЗНМ, англ. convolutional neural network, CNN, ConvNet) спочатку були розроблені для класифікації об’єктів зображень, а згодом вони успішно поширитись на обробку природної мови та текстових даних. Наприклад, CNN виконує функцію викладача шляхом обробки природньої мови людини, та з деякою помилкою трансформує вхідні осмислені речення у еквівалентні речення на іншій природній мові.

Згортка (англ. Convolution) – операція над парою цілочисельних матриць A (розміру $n_x \times n_y$) і B (розміру $m_x \times m_y$), результатом якої є матриця C = A * B з розміром $(n_x - m_x + 1) \times (n_y - m_y + 1)$ [16]. Кожен елемент результату обчислюється як скалярний твір матриці B і деякої підматриці A такого ж розміру (підматриця визначається положенням елемента в результаті). Математичний сенс представлений у якості формули 3.1.

$$C_{i.j} = \sum_{u=0}^{m_x-1} \sum_{v=0}^{y-1} A_{i+u,j+v} B_{u,v} \quad (3.1)$$

Логічний сенс згортки полягає у тому, що чим більше величина елемента згортки, тим більше ця частина матриці A була схожа на матрицю B (схожа в сенсі скалярного добутку). Тому матрицю A називають зображенням, а матрицю B називають фільтром або зразком.

Наукові дослідження домену допомагають узагальнити процес розробки багатошарової НМ та визначити послідовність наступних необхідних етапів [17].

1) Визначити вектор вхідних даних. Він повинен містити повністю всю достатню та необхідну для подальшого прогнозування інформацію.
2) У повній мірі визначити всі складові вихідного вектору, необхідні для повноцінної відповіді для поставленої задачі.
3) Вибрати оптимальну з точки зору завдання функцію активації нейронів.
4) Обрати архітектуру: тип та кількість шарів, і число нейронів у кожному з них.
5) Присвоїти початкові значення ваг і порогових рівнів. Для збереження прийнятної швидкості навчання, значення не повинні бути як великими, так і малими.
6) Провести навчання найкращим можливим чином, тобто вдало підібрати функцію втрат, за необхідності – оптимізатор, кількість епох навчання. Це налаштування до-зволить уникнути повільного навчання і, у подальшому, перенавчання. У результаті НМ зможе вирішувати подібні невідомі завдання.
7) Перевірити успішність функціонування мережі шляхом подання на її вхід задачі у вигляді знайомого вхідного вектору. Далі оцінити наданий моделлю

результат рішення у вигляді вихідного вектору на предмет справжності та дійсності.

Проведемо опис програмної середи та інструментів розробки. Jupyter Notebook – це основний інструмент для інтерактивної розробки та подання проектів в області наук про дані, який буде використовуватись у даній роботі. Блокнот об'єднує код і результат його роботи в єдиний документ, який може включати також візуалізацію, пояснюючий текст, математичні рівняння та інші мультимедійні дані [5].

Для розробки алгоритму та моделі використовується найбільш прийнятна для цього мова програмування – Python 3.8. Найбільш поширено у роботі використовуються такі бібліотеки:

– pandas – модуль для аналізу великих даних (обробка файлів типу csv);
– tensorflow – модуль для машинного навчання, розроблений компанією Google, з метою популяризації технології та спрощеного проведення експериментів у тренуванні НМ, зокрема виявлення об’єктів на зображеннях;
– keras – відкритий для загального користування модуль підтримки функцій, необхідних для розробки НМ. Може спільно працювати з tensorflow;
– sklearn – модуль для застосування, дослідження, тренування різноманітних алгоритмів класифікації, регресії та кластеризації.

У цьому дослідженні навчання моделі проводиться на публічному наборі даних CSIC 2012 Dataset. Він був штучно сформований за допомогою фреймворку Torpeda [18] для веб-сервісу електронної комерції та навмисно містить суттєву кількість зловмисного трафіку для тестування засобів захисту. Усі екземпляри даних Torpeda є загальнодоступними для спільноти та дослідників, як для позичення, так и для додавання нових контрольних даних. Запити складаються виключно із позначених HTTP-запитів, корисних для оцінки WAF (як на основі аномалій, так і на основі сигнатур).

CSIC 2012 Dataset містить усього 74 133 запитів, з яких: 49 311 є зловмисними, з різним типом ін’єкцій, 16 459 є аномальними (клас «anomalous») та 8 363 є звичайними (клас «normal»). Кількість запитів для ін’єкційних класів

наступна: 43 013 – різновиди SQL Injection, 4818 – різновиди XSS, 412 – Buffer Overflow, 41 – Format String, 74 – LDAPi (Lightweight Directory Access Protocol Injection), 451 – SSI (Server-Side Includes Injection), 175 – XPath, 327 – CRLF (Carriage Return Line Feed).

Звідси, конкретизуючи вимоги до нейронної моделі прийняття рішення, обмежимося існуванням наступних класів моделі: «SQLi», «XSS», «SSI», «BufferOverflow», «CRLFi», «XPath», «LDAPi», «FormatString», а також у разі неможливості вирішення конкретного типу атаки (або наявності атаки нульового дня) буде існувати загальне рішення – «anomalous», та у разі звичайного, незловмисного запиту буде існувати загальне рішення – «normal». Класифікація запитів у програмному коді продемонстрована на рисунку 3.1.

```
df.to_csv('./all.csv')
```

```
df['label'].value_counts()
```

```
SQLi              43013
anomalous         16459
normal             8363
XSS                4818
SSI                 451
BufferOverflow      412
CRLFi               327
XPath               175
LDAPi                74
FormatString         41
Name: label, dtype: int64
```

Рисунок 3.1 – Класифікація запитів набора даних

Набір даних був представлений окремим переліком файлів, які були відсортовані за папками: anomalies, attacks, normals. Попередня обробка перш за все включала у себе комбінування усіх файлів у єдиний файл.

Отже, dataset був об'єднаний у Excel таблицю (тип файлу csv), кожен рядок якої підлягає виразу:

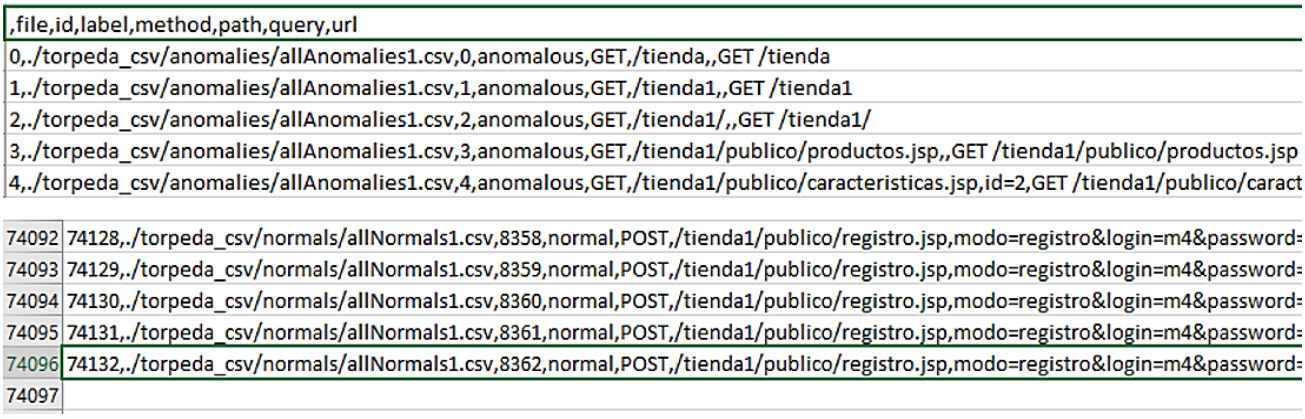
– file (назва вихідного файлу), id (номер запиту у певному класі);
– label (відмітка класу);
– method (методи HTTP-протоколу – GET або POST);

– path (частина URL, шлях до сторінки, яка відображається клієнту);

– query (набір параметрів, що передаються у запиті);

– url (повне посилання – path + query).

Отриманий набір даних представлений на рисунку 3.2.

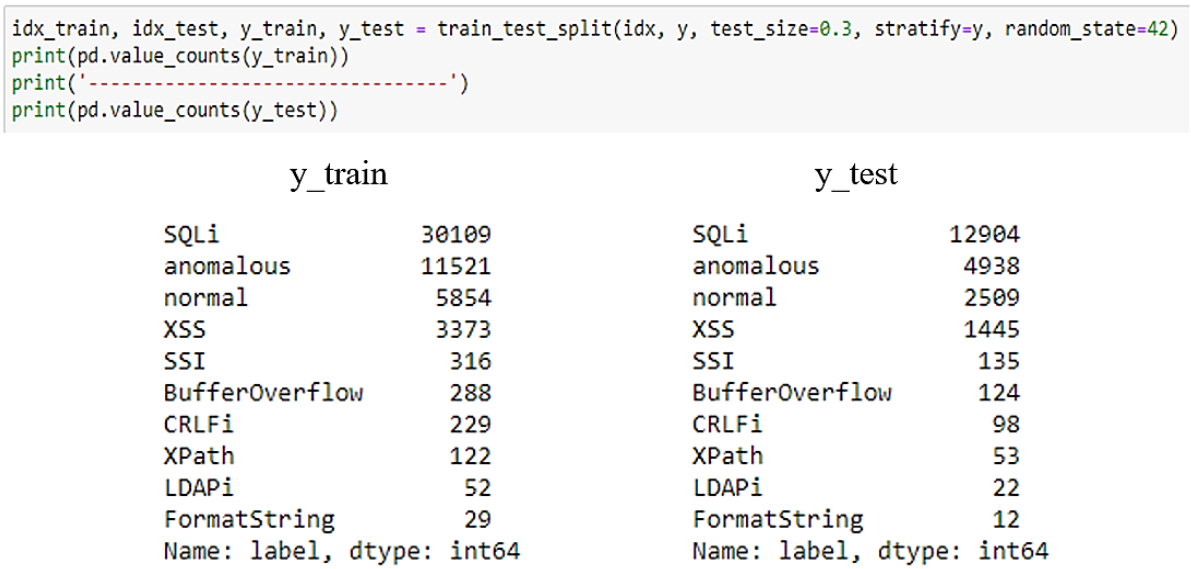


	,file,id,label,method,path,query,url
0	0,./torpeda_csv/anomalies/allAnomalies1.csv,0,anomalous,GET,/tienda,,GET /tienda
1	1,./torpeda_csv/anomalies/allAnomalies1.csv,1,anomalous,GET,/tienda1,,GET /tienda1
2	2,./torpeda_csv/anomalies/allAnomalies1.csv,2,anomalous,GET,/tienda1/,,GET /tienda1/
3	3,./torpeda_csv/anomalies/allAnomalies1.csv,3,anomalous,GET,/tienda1/publico/productos.jsp,,GET /tienda1/publico/productos.jsp
4	4,./torpeda_csv/anomalies/allAnomalies1.csv,4,anomalous,GET,/tienda1/publico/caracteristicas.jsp,id=2,GET /tienda1/publico/caract
74092	74128,./torpeda_csv/normals/allNormals1.csv,8358,normal,POST,/tienda1/publico/registro.jsp,modo=registro&login=m4&password=
74093	74129,./torpeda_csv/normals/allNormals1.csv,8359,normal,POST,/tienda1/publico/registro.jsp,modo=registro&login=m4&password=
74094	74130,./torpeda_csv/normals/allNormals1.csv,8360,normal,POST,/tienda1/publico/registro.jsp,modo=registro&login=m4&password=
74095	74131,./torpeda_csv/normals/allNormals1.csv,8361,normal,POST,/tienda1/publico/registro.jsp,modo=registro&login=m4&password=
74096	74132,./torpeda_csv/normals/allNormals1.csv,8362,normal,POST,/tienda1/publico/registro.jsp,modo=registro&login=m4&password=
74097	

Рисунок 3.2 – Початок і кінець об’єднаного набору даних

При роботі з машинним навчанням, дані необхідно поділити як мінімум на 2 нерівні, збалансовані частини, що не перекреслюються: навчання і тестування. Попередня валідація після навчання є необов’язковим етапом.

Розбиття набору даних має важливе значення для неупередженої оцінки ефективності прогнозування. Поділ набору даних наведений на рисунку 3.3.



```
idx_train, idx_test, y_train, y_test = train_test_split(idx, y, test_size=0.3, stratify=y, random_state=42)
print(pd.value_counts(y_train))
print('--------------------------------')
print(pd.value_counts(y_test))
```

y_train

```
SQLi              30109
anomalous         11521
normal             5854
XSS                3373
SSI                 316
BufferOverflow      288
CRLFi               229
XPath               122
LDAPi                52
FormatString         29
Name: label, dtype: int64
```

y_test

```
SQLi              12904
anomalous          4938
normal             2509
XSS                1445
SSI                 135
BufferOverflow      124
CRLFi                98
XPath                53
LDAPi                22
FormatString         12
Name: label, dtype: int64
```

Рисунок 3.3 – Поділ набору даних: навчання і тестування

Функція train_test_split() з бібіліотеки sklearn надає змогу рівномірно поділити випадково перемішений (параметр random_state=42) набір даних у

відношенні: вибірка навчання – 70%, вибірка тестування – 30% (параметр test_side=0.3). Параметр stratify = y дозволяє приблизно зберегти частку значень «y» у навчальних та тестових вибірках, у чому можна переконатися завдяки розгляду кількості значень y_train та y_test [5].

Розбиття набору даних також може мати важливе значення для виявлення того, що розроблена модель страждає від однієї з двох дуже поширених проблем, які називаються недостатнім або надмірним навчанням.

Недостатнє навчання зазвичай свідчить про те, що модель не володіє достатніми зв'язками між даними. Наприклад, це може трапитися у разі надання нелінійний даних на вхід НМ з лінійною функцію активації. Проблема виражається у низьких метриках якості під час навчання і тестування.

Перенавчання свідчить про те, НМ надто складно побудована і тому аналізує не тільки реальні зв'язки між даними. Може виникнути ситуація, де НМ витягує занадто багато інформації з окремих випадків, «забуваючи» відповідну інформацію загального випадку.

Наступним кроком буде проведення маніпуляцій з частиною набору даних для навчання, які носять назву обробки природної мови (Natural Language Processing, NLP). Мета попередньої обробки (preprocessing) полягає у переведенні текстових символів у формат, з яким у подальшому буде працювати НМ. Сьогодні NLP застосовується в багатьох сферах, в тому числі в голосових помічниках, автоматичних перекладах тексту і фільтрації тексту. У даному випадку реалізація NLP представлена модулем keras.preprocessing.text [5].

Доступно багато підходів до попередньої обробки, які відрізняються в залежності від вхідних даних та необхідної вихідної форми:

– Очищення даних (видалення або заміна чисел, зайвих пробілів, табуляції, пунктуації, спеціальних символів, стоп-слів).
– Редагування даних (наприклад, переведення у верхній\нижній регістр).
– Скорочення даних.
– Токенізація (зазвичай реалізується на основі регулярних виразів).
– Векторизація.

Клас keras.preprocessing.text.Tokenizer є реалізацією методів векторизації текстових даних, тобто методів репрезентації тексту у числовому поданні. Таким чином, слова або фрази з кожного вхідного текстового значення відображаються у відповідний вектор дійсних чисел зі спільного для усіх запитів словника.

За замовчуванням вся пунктуація видаляється і текст перетворюється в послідовність слів (слова можуть включати символ ' – одиночний апостроф). Потім ці послідовності розбиваються на списки з токенів. Далі можна виконувати векторизацію або індексацію.

Спираючись на діаграму класів (див. рис. 1.4), був створений клас DataProcessor, який обробляє екземпляри класу UrlRequest.

У даному класі складався словник з лексем (див. рис. 3.4), для чого був створений об'єкт Tokenizer з параметрами [20]:

– filters: регулярний вираз з символами, які будуть виключені з тексту. За замовчуванням виключаються всі розділові знаки, символи табуляції і розриву рядка, виключаючи символ ' (одиночний апостроф);
– char_level: якщо вказано значення True, тоді кожен символ слова буде розглядатися як токен.

```
# Character-level word segmentation, fitting on the training set
tokenizer = Tokenizer(filters='\t\n', char_level=True)
tokenizer.fit_on_texts(url_train)
# Build a dictionary and save
num_words = len(tokenizer.word_index)+1
vocab = tokenizer.word_index
print("The size of the dictionary is %d" % num_words)
print("dictionary: ")
print(vocab)
with open("./tokenizer/vocab.json", 'w') as f:
    json.dump(vocab, f, ensure_ascii=False)
```

```
The size of the dictionary is 72
dictionary:
{'%': 1, '2': 2, 'c': 3, '0': 4, 'i': 5, 'e': 6, 'r': 7, 'o': 8, '3': 9, 'a': 1
0, 'n': 11, '=': 12, 'l': 13, 'm': 14, '&': 15, 'd': 16, '1': 17, '7': 18, 't':
19, 's': 20, 'p': 21, '5': 22, '6': 23, 'u': 24, '9': 25, '8': 26, 'b': 27,
'/': 28, ',': 29, '4': 30, 'g': 31, '.': 32, 'f': 33, 'j': 34, 'w': 35, ' ': 3
6, '?': 37, 'v': 38, 'h': 39, 'z': 40, 'x': 41, '+': 42, 'y': 43, '-': 44, 'k':
45, '#': 46, 'q': 47, ';': 48, '<': 49, '>': 50, '"': 51, '_': 52, '@': 53,
'*': 54, ':': 55, '(': 56, ')': 57, "'": 58, '!': 59, '[': 60, ']': 61, '{': 6
2, '}': 63, '`': 64, '\r': 65, '$': 66, '~': 67, '|': 68, '\\': 69, '^': 70,
'\n': 71}
```

Рисунок 3.4 – Складання словника шляхом сегментації слів

Метод fit_on_texts() оновлює внутрішню лексику на основі списку текстів. Цей метод створює словниковий індекс на основі частоти токенів від найбільшої до найменшої. Окрім цього, зазначимо, що найчастіше зустрічається у запитах саме символ «%», а найрідшим є символ «\n». Атрибут word_index містить таблицю для словника, де кожен елемент має унікальне призначене число [20].

На рисунку 3.5 показана програмна реалізація, де метод texts_to_sequences() перетворює кожен символ кожного UrlRequest у послідовність цілих чисел. Концептуально цей процес відповідає методу processUrl(), який відображений у діаграмі класів (див. рис. 1.4) . Також, клас DataProcessor містить атрибути seq_train і seq_test, які у діаграмі класів були відображені як dataTrain, dataTest відповідно.

```
# After observation, define the maximum length of url to be 600
max_len = 600
# Convert text to sequence
seq_train = tokenizer.texts_to_sequences(url_train)
seq_test = tokenizer.texts_to_sequences(url_test)
# Unify the length of the sequence to max_len
X_train = np.asarray(sequence.pad_sequences(seq_train, maxlen=max_len))
X_test = np.asarray(sequence.pad_sequences(seq_test, maxlen=max_len))
# Convert tags to one-hot
Y_train = np.asarray([get_one_hot_value(l) for l in label_train])
Y_test =  np.asarray([get_one_hot_value(l) for l in label_test])
```

Рисунок 3.5 – Перетворення запитів у вектори

```
Before vectorization:
url: POST /tienda1/publico/pagar.jsp?precio=85&B1=Pasar%2Bpor%2Bcaja%27%2C%270%
27%2C%270%27%29%3Bwaitfor+delay+%270%3A0%3A15%27%3B--
label: SQLi
After vectorization:
url_vec: [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 21  8 20 19 36 28 19
  5  6 11 16 10 17 28 21 24 27 13  5  3  8 28 21 10 31 10  7 32 34 20 21
 37 21  7  6  3  5  8 12 26 22 15 27 17 12 21 10 20 10  7  1  2 27 21  8
  7  1  2 27  3 10 34 10  1  2 18  1  2  3  1  2 18  4  1  2 18  1  2  3
  1  2 18  4  1  2 18  1  2 25  1  9 27 35 10  5 19 33  8  7 42 16  6 13
 10 43 42  1  2 18  4  1  9 10  4  1  9 10 17 22  1  2 18  1  9 27 44 44]
```

Рисунок 3.6 – Результат векторизації

На рисунку 3.6 показано, що після векторизації більшість чисел дорівнює нулю для окремого запиту. Можна пояснити це тим, що для кожного URL-посилання статично виділена максимальна довжина 600 символів. Видно, що перші 4 цифри після низки нулів, які представлені на рисунку, згідно отриманому вище словнику, представляють слово «post».

Розробимо клас NeuralNetwork згідно діаграми класів (рис. 1.4). Ключовий атрибут цього класу – model, отже, виберемо програмну модель нейронної мережі. У Keras доступні два основних типи моделей: Sequential і Model, причому остання вимагає підтримки функціонального API. Перша Sequential модель є послідовною низкою доступних шарів. Якщо потрібно створити довільну організацію шарів, можна використати розширений API Keras, але у даному дослідженні достатньо Sequential моделі [20].

Архітектура кожної згорткової нейронної мережі (ЗНМ) обов'язково включає в себе наступні структурні шари у відповідній послідовності: згорткові шари (англ. Convolutional layer), пулінгові шари (англ. Pooling layer) і повнозв'язні шари (англ. Fully-connected layer). Нижче наведений короткий опис сенсу введення кожного окремого шару моделі.

Шар Embedding можна представити у вигляді простого матричного множення, яке перетворює натуральні числа (індекси) на щільні вектори фіксованого розміру, наприклад [[4], [20]] → [[0.25, 0.1], [0.6, -0.2]].

Модель повинна знати розмірність вхідного масиву даних. Тому перший шар в Sequential моделі (і тільки перший, тому що наступні шари автоматично отримують цю інформацію з попереднього шару) повинен отримувати інформацію про розмірність вхідного масиву [5].

Згортковий шар – ключова складова ЗНМ. Шар згортки є сукупністю фільтів для кожного каналу, ядро згортки якого обробляє попередній шар за фрагментами (підсумовуючи результати матричного твору для кожного фрагмента). Вагові коефіцієнти ядра згортки (невеликої матриці) невідомі і встановлюються в процесі навчання. Conv1D – це шар, який створює згорткове ядро за одним просторовим (або часовим) вимірюванням.

Шар Pooling (Об'єднання) зменшує кількість параметрів, для подальшого навчання та, відповідно, обсяг обчислень мережі. Рівень об'єднання зберігає тільки найбільш суттєві ознаки, згенеровані попередньо шаром згортки [1]. Перетворення відбуваються для кожного окремого квадрата, кожен з яких скорочується в один піксель, який має максимальне значення. Частіш за все мається на увазі операція max() у шарі MaxPooling (див. рис. 3.9).

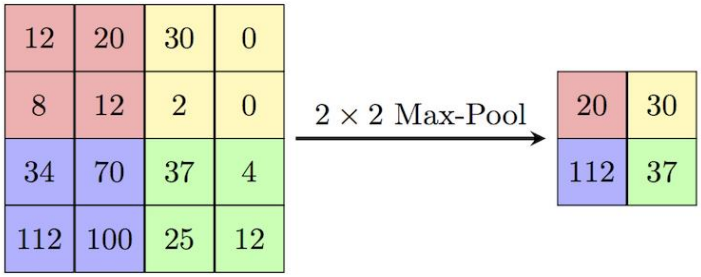


Рисунок 3.7 – Операція max-pooling

Щільний шар (Dense) є повністю пов'язаним із попереднім шаром. Нейрони щільного шару здійснюють матрично-векторне множення. Dense вводиться наприкінці, для зміни розмірності виходу (у даному випадку для 10 класів кінцева розмірність шару має бути 10) та реалізує операцію output = activation (dot (input, kernel) + bias) [1].

Щодо вибору функцій активації, були застосовані дві найпопулярніші. Rectified Linear Unit (ReLU) – нелінійна функція активації, яка повертає 0 у разі негативного аргументу, а при позитивному аргументі – його чисельне значення залишається незмінним на виході [19]. ReLU відповідає математичному виразу 3.2.

$$f(z) = \max(0, z) \tag{3.2}$$

Графік функцій активації ReLU зображений на рисунку 3.8.

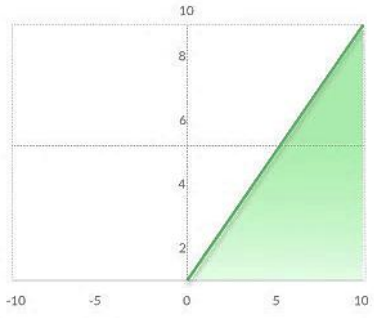


Рисунок 3.8 – Графік функцій активації ReLU

Функцію активації ReLU слід використовувати, якщо немає особливих вимог для вихідного значення нейрона, наприклад, при необмеженій області визначення. Але якщо після навчання моделі результати вийшли не оптимальні, то у наступних версіях нейронної мережі варто перейти до інших функцій, які можуть дати кращий результат [5].

Функція активації Softmax – це узагальнення логістичної функції для багатовимірного випадку. Функція перетворює вектор z розмірності n у вектор σ тієї ж розмірності, де кожна координата σ_i отриманого вектора представлена дійсним числом в інтервалі [0,1] і сума координат дорівнює 1 [21]. Функція Softmax застосовується в машинному навчанні для задач класифікації, коли кількість можливих класів більше двох (для двох класів використовується логістична функція). Координати отриманого вектору при цьому трактуються як ймовірності того, що об'єкт належить до класу і.

Отже, перед підготовкою моделі необхідно налаштувати процес навчання. Метод compile має три основні аргументи:

1) Функція обрахування помилок. Функція втрат – це показник якості прогнозування, процент невдалих передбачень моделі відносно очікуваних результатів. Бажано звести неправдиві передбачення до мінімуму. Функція може бути строковим ідентифікатором існуючої функції втрат (наприклад, categorical_crossentropy або mse) або може бути цільовою функцією.

Для задач машинного навчання в переважній більшості випадків використовується крос-ентропія (перехресна ентропія). Будь то категорична, розріджена або бінарна крос-ентропія, вона є стандартним вибором для високопродуктивних нейронних мереж [22].

Ентропія випадкової величини X — це рівень невизначеності, притаманний можливому результату змінних. Перехресна ентропія вимірює різницю між виявленим розподілом P ймовірностей моделі класифікації машинного навчання та прогнозованим розподілом Q. Крос-ентропія має переваги:

– Метрики, засновані на accuracy, є дуже чутливими до розподілу даних в наборі для навчання. Для них не враховується статистична достовірність, і

вони не є стійкими відносно до різних атрибутів даних, що може привести до помилкових результатів. Тобто ці методи є досить грубими.

– Крос-ентропія враховує інформаційне наповнення, і тому вона більш динамічна і стійка, ніж метрики, які просто перевіряють, чи відзначені всі пункти відстеження. Прогнози і наміри в ній представлені розподілами, а не переліком статичних питань.
– Крос\ентропія працює на основі розподілів і ймовірностей, і тому добре поєднується з сигмоїдою і SoftMax (навіть у випадку, коли вони використані для нейронів в останньому шарі), допомагаючи зменшити проблему зникаючого градієнта.

2) Оптимізатор – це алгоритм, який вирішує поширену проблему надто повільного навчання моделі. Він дозволяє зменшити час та скоригувати якість навчання моделі також шляхом певної незначної зміни ваг нейронів.

Зосередимо вибір на оптимізаторі Adaptive Moment Estimation (Adam) — це алгоритм для техніки оптимізації градієнтного спуску, що базується на адаптивній оцінці імпульсів першого та другого порядку. Метод дійсно ефективний при роботі з задачею, що включає велику кількість даних або параметрів. Згідно дослідженням, алгоритм потребує порівняно менше пам'яті та є ефективним [23].

Оптимізатор Адама включає дві технології градієнтного спуску, а саме:

– Оптимізатор імпульсу (англ. Momentum), який працює шляхом врахування «експоненціальної зваженої середини» градієнтів. Використання середніх значень змушує алгоритм наближатися до мінімумів швидше.
– Середньоквадратичне поширення (RMSProp), у якому швидкості навчання налаштовуються для кожного параметра, які, у свою чергу, адаптовані на основі середнього значення останніх величин градієнтів для ваги. Алгоритм є модифікованою версією метода AdaGrad. Разом з паралельним розширенням AdaGrad – AdaDelta – обидва алгоритми були створені для вирішення проблеми безконтрольного накопичення квадратів градієнтів, що зрештою призводило до зупинки процесу навчання.

3) Список метрик. Метрику можна вказати для метода, користуючись строковим ідентифікатором відповідної метрики або спеціальною метричної функцією. Для будь-якої проблеми класифікації за замовчуванням встановлюється metrics = ['accuracy'].

На рисунку 3.9 наведені усі вищеописані етапи створення моделі з вказанням усіх параметрів у вигляді програмного коду.

```
# Build a network
model = Sequential()
model.add(layers.Embedding(num_words, 64, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Рисунок 3.9 – Програмна реалізація створення Sequential моделі

На рисунку 3.10 зображений згенерований програмно звіт про отриману архітектуру моделі, кількість нейронів на кожному шарі та кількість вхідних параметрів. Також представлена візуалізація шарів відповідно до їх розмірів.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 600, 64)           4608
_________________________________________________________________
conv1d (Conv1D)              (None, 594, 32)           14368
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 118, 32)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 112, 32)           7200
_________________________________________________________________
global_max_pooling1d (Global (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 10)                330
=================================================================
Total params: 26,506
Trainable params: 26,506
Non-trainable params: 0
_________________________________________________________________
```

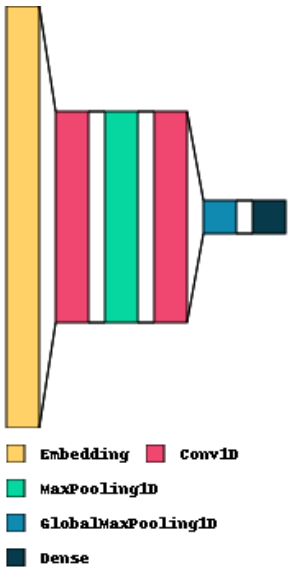


Рисунок 3.10 – Звіт про архітектуру моделі

На рисунку 3.11 представлена робота кожного шару згорткової нейронної моделі, тобто як з вхідних слів першого шару модель поетапно проводить класифікацію, обираючи найбільш помітні ознаки того чи іншого класу.

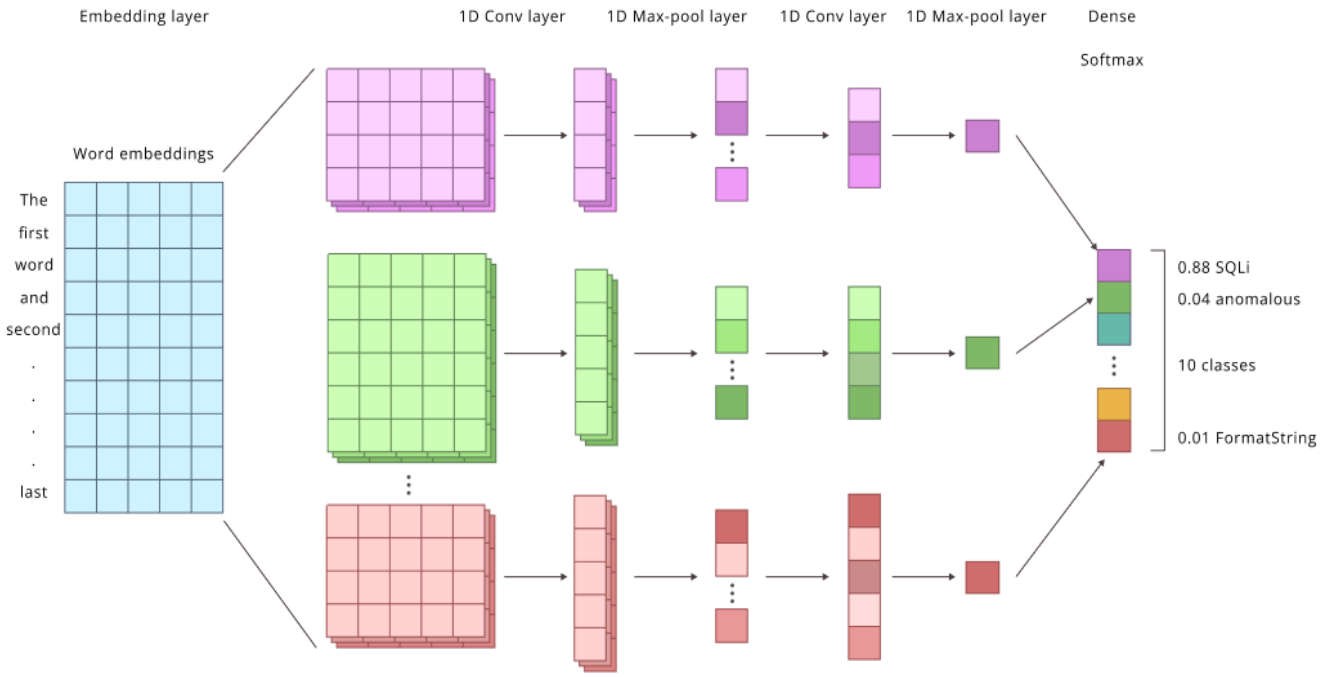


Рисунок 3.11 – Функціонування нейронної мережі

Навчання моделі відбувалося на ноутбуку Asus з процесором Intel Core i5-6198DU CPU 2.3GHz. Для навчання за допомогою метода fit() у якості параметру оптимально було обрано 6 епох, кожна з яких тривала приблизно 5 секунд. Епоха є однією ітерацією в процесі навчання, що включає пред'явлення всіх прикладів з навчальної множини і перевірку якості навчання на контрольній множині. У кожній наступній епосі підвищується точність та зменшується величина помилки.

```
# training
model.fit(X_train, Y_train, epochs=6, batch_size=128, callbacks=[tb_callback])
```

```
Epoch 1/6
  2/406 [..............................] - ETA: 8:56 - loss: 2.2095 - accuracy: 0.5898WARNING
_end) is slow compared to the batch update (1.191647). Check your callbacks.
406/406 [==============================] - 87s 215ms/step - loss: 0.2848 - accuracy: 0.9214
Epoch 2/6
406/406 [==============================] - 88s 216ms/step - loss: 0.0345 - accuracy: 0.9896
Epoch 3/6
406/406 [==============================] - 82s 203ms/step - loss: 0.0163 - accuracy: 0.9945
Epoch 4/6
406/406 [==============================] - 80s 198ms/step - loss: 0.0133 - accuracy: 0.9951
Epoch 5/6
406/406 [==============================] - 86s 211ms/step - loss: 0.0115 - accuracy: 0.9959
Epoch 6/6
406/406 [==============================] - 89s 219ms/step - loss: 0.0101 - accuracy: 0.9959
```

Рисунок 3.12 – Навчання моделі

3.2 Вибір нейронної моделі

Варто враховувати, що необхідно створити нейронну мережу з найкращою конфігурацією параметрів, для цього було розглянуто різні варіанти класифікаторів нейронної мережі (Random Forest, K-Neighboors, Decisions Tree) та зроблено аналіз.

Спочатку проведемо верифікацію поточної згорткової моделі. Після навчання згорткової НМ можна отримати данні про успішність моделі за допомогою метода evaluate(). На рисунку 3.13 метод відповідно повернув дві метрики: величина помилки (loss) та точність моделі (accuracy).

```
# Evaluation model
model.evaluate(X_test, Y_test, batch_size=128)
```

```
174/174 [==============================] - 8s 47ms/step - loss: 0.0139 - accuracy: 0.9956
[0.013854621909558773, 0.9956384897232056]
```

Рисунок 3.13 – Втрати та точність моделі після навчання

У завданнях машинного навчання для оцінки якості моделей і порівняння різних алгоритмів навчання використовуються метрики. Перед розглядом власно метрик необхідно ввести важливу концепцію для опису цих метрик в термінах помилок класифікації.

У галузі машинного навчання, й зокрема для проблеми класифікації, матриця невідповідностей (англ. confusion matrix), або матриця помилок (англ. error matrix), — це метод підсумовування продуктивності алгоритму класифікації. Данна матриця репрезентує порівняне прогнозоване значення з фактичним значенням (див. рисунок 8). Кожен рядок цієї таблиці є екземпляром фактичного класу змінної, тоді як кожен стовпчик є екземпляром прогнозованого класу.

Для кращого розуміння введемо наступні терміни:

– Позитивний стан (П) – число справжніх позитивних передбачень.
– Негативний стан (Н) – число справжніх негативних передбачень.
– True Positive (істинно-позитивне рішення, TP) – передбачення співпадає з дійсністю, відповідь позитивна, що було передбачено НМ.
– False Positive (хибно-позитивне рішення, FP) – помилка I роду, НМ передбачила позитивний результат, а насправді він негативний.

– False Negative (хибно-негативне рішення, FN) – помилка II роду – НМ надала негативну відповідь, але насправді вона позитивна.
– True Negative (істинно-негативне рішення, TN) – результат негативний, прогноз НМ збігся з реальністю.

Accuracy – це частка правильних відповідей алгоритму (формула 3.3):

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (3.3)$$

Для оцінки якості алгоритму для окремого класу використовуються метрики precision (точність, формула 3.4) і recall (повнота, формула 3.5).

$$precision = \frac{TP}{TP+FP} \quad (3.4),$$

$$recall = \frac{TP}{TP+FN} \quad (3.5).$$

Precision можна інтерпретувати як частку запитів, названих класифікатором позитивними, і вони дійсно позитивні, а recall показує, яку частину запитів позитивного класу з усіх запитів позитивного класу знайшла модель.

Також можна різними способами поєднати precision і recall в комбіновану метрику. Наведемо, наприклад, формулу для обрахування F-міри на основі precision і recall (формула 3.6):

$$F_\beta = (1+\beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall} \quad (3.6).$$

F-міра досягає свого максимуму, коли точність і повнота дорівнюють одиниці, і досягає свого мінімуму, якщо один з параметрів близький до нуля.

Попередні програмні заходи для отримання метрик якості показані на рис. 3.14. Усі використані методи їх отримання знаходяться у стандартному API Keras.

На рисунку 3.15 та 3.16 зображена надана програмою статистика щодо метрик успішності машинного навчання відповідно: матриця помилок та набір параметрів. З матриці помилок видно, що НМ є достатньо ефективною та помилки першого та другого роду майже для усіх класів не виникають (перетин стовпця та рядка дорівнює 0).

```
Y_test_pred = model.predict(X_test)
label_test_pred = props_to_labels(Y_test_pred)
print("Confusion matrix: ")
print(confusion_matrix(label_test, label_test_pred, labels=labels_type))
print("f1-score:")
print(f1_score(label_test, label_test_pred, labels=labels_type, average='micro'))
print("acc-score:")
print(accuracy_score(label_test, label_test_pred))
print("recall-score:")
print(recall_score(label_test, label_test_pred, labels=labels_type, average='micro'))
print("classification report:")
print(classification_report(label_test, label_test_pred, labels=labels_type))

f1_dict["CNN"] = f1_score(label_test, label_test_pred, average='micro')
precision_dict["CNN"] = precision_score(label_test, label_test_pred, labels=labels_type, average='micro')
recall_dict["CNN"] = recall_score(label_test, label_test_pred, labels=labels_type, average='micro')
accuracy_dict['CNN'] = accuracy_score(label_test, label_test_pred)
```

Рисунок 3.14 – Програмний код отримання метрик якості CNN

Загальний висновок наступний: у контексті набору даних Torpeda, CNN є вкрай продуктивною та передбачає наявність практично усіх розглянутих атак.

```
[[12903     0     0     0     0     0     0     1     0     0]
 [    3  4919    16     0     0     0     0     0     0     0]
 [    0    17  2492     0     0     0     0     0     0     0]
 [    6     0     0  1437     0     0     0     1     0     1]
 [    0     6     3     0   125     0     0     0     0     1]
 [    0     0     0     0     2   122     0     0     0     0]
 [    0     0     0     0     0     0    98     0     0     0]
 [   31     0     0     2     4     0     0    15     0     1]
 [    0     0     0     0     0     0     0     0    21     1]
 [    0     1     0     0     0     0     0     0     0    11]]
```

Рисунок 3.15 – Матриця невідповідностей (помилок)

	precision	recall	f1-score	support
SQLi	1.00	1.00	1.00	12904
anomalous	1.00	1.00	1.00	4938
normal	0.99	0.99	0.99	2509
XSS	1.00	0.99	1.00	1445
SSI	0.95	0.93	0.94	135
BufferOverflow	1.00	0.98	0.99	124
CRLFi	1.00	1.00	1.00	98
XPath	0.88	0.28	0.43	53
LDAPi	1.00	0.95	0.98	22
FormatString	0.73	0.92	0.81	12
accuracy			1.00	22240
macro avg	0.96	0.90	0.91	22240
weighted avg	1.00	1.00	1.00	22240

Рисунок 3.16 – Метрики якості НМ

Отримавши високі показники точності прогнозування згорткової моделі, наступним шагом буде порівняти метрики з іншими класифікаторами.

Random Forest – це метод машинного навчання для класифікації алгоритмів. Він складається з кількох окремих дерев рішень, які спираються на випадкові особливості та навчання даних, щоб досягти розумного припущення, яке має більше довіри, ніж одне дерево рішень. Усі дерева рішень у довільному лісі є окремими моделями. Кожна з них використовує підмножину випадкових ознак для передбачення мети, і всі ці цілі, що передбачаються, накопичуються разом для передбачення більш точної мети. Формула підсумкового класифікатора a(x) визначається як (3.3), де N – кількість дерев; i – лічильник для дерев; b – вирішальне дерево; x – згенерована вибірка даних.

$$a(x) = \frac{1}{N}\sum_{i=1}^{N} b_i(x) \tag{3.7}$$

На рис. 3.17 наведений програмний код використання класифікатору Random Forest у рамках даної задачі. Спочатку класифікатор навчається (метод fit()), далі він надає прогнозування (метод predict()), яке потім порівнюється з відомим тестовим набором міток класів (Y_test). Від виду порівняння залежить отримана на виході метрика моделі. У даному випадку для усіх класифікаторів розглядаються f-score, precision, recall, accurancy.

```
# RANDOM FOREST
```

```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, Y_train)
y_pred = rf_clf.predict(X_test)
print(f"Accuracy of Random Forest on test set : {accuracy_score(y_pred, Y_test)}")
print(f"F1 Score of Random Forest on test set : {f1_score(y_pred, Y_test, average='micro')}")

# Updates model score to f1_dict
f1_dict["RandomForest"] = f1_score(y_pred, Y_test, average='micro')
precision_dict["RandomForest"] = precision_score(y_pred, Y_test, average='micro')
recall_dict["RandomForest"] = recall_score(y_pred, Y_test, average='micro')
accuracy_dict['RandomForest'] = accuracy_score(y_pred, Y_test)
```

```
Accuracy of Random Forest on test set : 0.9830035971223021
F1 Score of Random Forest on test set : 0.9879343847440011
```

Рисунок 3.17 – Метрики класифікатора Random Forest

K-Neighbors (KNN) – це тип алгоритму машинного навчання, який використовується як для регресії, так і для класифікації. Класифікатор KNN розраховує ймовірність того, що тестові дані належать до K класів навчальних

даних, і результативним класом буде вибраний клас із найвищою ймовірністю співпадіння. Алгоритм KNN намагається передбачити правильний клас для тестових даних, обчислюючи евклідову відстань між тестовими даними та всіма точками навчання. Потім вибираються K найближчих сусідів відповідно до розрахованої евклідової відстані. Серед цих k сусідів підраховується кількість точок даних у кожній категорії. Далі призначаються нові точки даних категорії, для якої кількість сусідів максимальна. На рис. 3.18 наведений програмний код використання класифікатору K-Neighboors.

```
# K Neighbor sClassifier
```

```
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, Y_train)
y_pred = knn_clf.predict(X_test)
print(f"Accuracy of K Neighbors on test set : {accuracy_score(y_pred, Y_test)}")
print(f"F1 Score of K Neighbors on test set : {f1_score(y_pred, Y_test, average='micro')}")

# Updates model score to f1_dict
f1_dict["KNN"] = f1_score(y_pred, Y_test, average='micro')
precision_dict["KNN"] = precision_score(y_pred, Y_test, average='micro')
recall_dict["KNN"] = recall_score(y_pred, Y_test, average='micro')
accuracy_dict['KNN'] = accuracy_score(y_pred, Y_test)
```

```
Accuracy of K Neighbors on test set : 0.9736510791366907
F1 Score of K Neighbors on test set : 0.9767693626234832
```

Рисунок 3.18 – Метрики класифікатора K-Neighbors

Дерева рішень (англ. Decision Tree) є інструментом передбачуваної аналітики, що дозволяє вирішувати завдання класифікації. Вони є ієрархічними деревоподібними структурами, які складаються з вирішальних правил виду «якщо ..., то ...». Правила автоматично генеруються в процесі навчання на навчальній множині. Метою використання Decision Tree є створення навчальної моделі, яка може використовуватися для прогнозування класу або значення цільової змінної шляхом вивчення простих правил прийняття рішень, виведених із попередніх даних (навчальних даних). У деревах рішень прогнозування мітки класу для запису починається з кореня дерева. Алгоритм порівнює значення кореневого атрибута з атрибутом вхідного запису. На основі порівняння алгоритм слідує за гілкою, що відповідає цьому значенню, і переходить до наступного вузла. Він продовжує процес, доки не досягне листового вузла дерева.

```
# DECISION TREES
```

```
DT = tree.DecisionTreeClassifier()
DT.fit(X_train, Y_train)
y_pred = DT.predict(X_test)
print(f"Accuracy of Decision Tree on test set : {accuracy_score(y_pred, Y_test)}")
print(f"F1 Score of Decision Tree on test set : {f1_score(y_pred, Y_test, average='micro')}")

# Updates model score to f1_dict
f1_dict["DecisionTree"] = f1_score(y_pred, Y_test, average='micro')
precision_dict["DecisionTree"] = precision_score(y_pred, Y_test, average='micro')
recall_dict["DecisionTree"] = recall_score(y_pred, Y_test, average='micro')
accuracy_dict['DecisionTree'] = accuracy_score(y_pred, Y_test)
```

```
Accuracy of Naive Bayes on test set : 0.983453237410072
F1 Score of Naive Bayes on test set : 0.9841612670986322
```

Рисунок 3.19 – Метрики класифікатора Decision Trees

На рисунку 3.20 представлена зведена таблиця метрик точності розглянутих класифікаторів. На рисунку 3.21 приведений програмно згенерований графік на основі таблиці для кращої візуалізації результатів.

```
keys2 = f1_dict, precision_dict, recall_dict, accuracy_dict
metrics = ['F1_Score', 'Precision', 'Recall', 'Accuracy']
data = pd.DataFrame(keys2)
data.index = metrics
print(data)
```

```
           RandomForest       KNN  DecisionTree       CNN
F1_Score       0.987934  0.976769      0.984161  0.993490
Precision      0.983004  0.973651      0.983453  0.994649
Recall         0.992915  0.979908      0.984870  0.998734
Accuracy       0.983004  0.973651      0.983453  0.994649
```

Рисунок 3.20 – Порівняння метрик класифікаторів

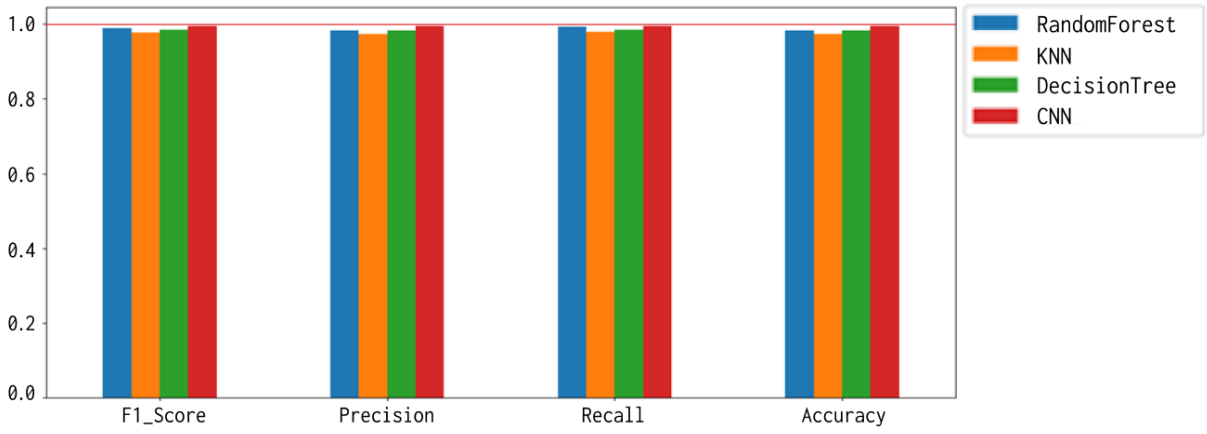


Рисунок 3.21 – Графік порівняння метрик класифікаторів

Піддаючи аналізу отримані метрики, можна зауважити, що загорткова модель найкраще впорюється з поставленою задачею класифікації. Також видно, що інші класифікатори прогнозують класи дуже точно. Отже, основною моделлю

можна обрати згорткову НМ, але при верифікації моделі у реальних умовах неможливо передбачити поведінку різних моделей, тому варто зберегти усі з них.

Проведемо верифікацію CNN після навчання. Тестування можна вважати завершеним, коли кількість тестових запитів покрила точність показника якості навчання. Якщо показник дорівнює одній тисячній, то щоб повноцінно підтвердити її потрібно як мінімум 1000 тестових запитів.

Тестування якості НМ є важким завданням з точки зору апаратної частини, адже воно вимагає продуктивного обладнання і найближчих до реальних умов тестових даних. Тобто для реального застосування ЕС для web-додатків потрібен певний час для налаштування під дані дійсного трафіку. Стають можливими ризики погіршення точності НМ, і у цьому разі необхідно впроваджувати додаткові заходи оптимізації, налаштування вхідних параметрів, або навіть перенавчання.

Підготовка до перевірки НМ включала дії: 22240 унікальних тестових запитів були розміщені у файлі url_test.txt (рис. 3.22), а відомі відповідні до них мітки класів у файлі label_test.txt (рис. 3.23); на вхід нейронної моделі подавався url_test.txt, а результат детектування кібератак (моделлю) ставився у відповідність до label_test.txt, тобто НМ не мала доступу до дійсних міток класу на цей раз; вручну проводилось порівняння фактичних показників із передбаченими.

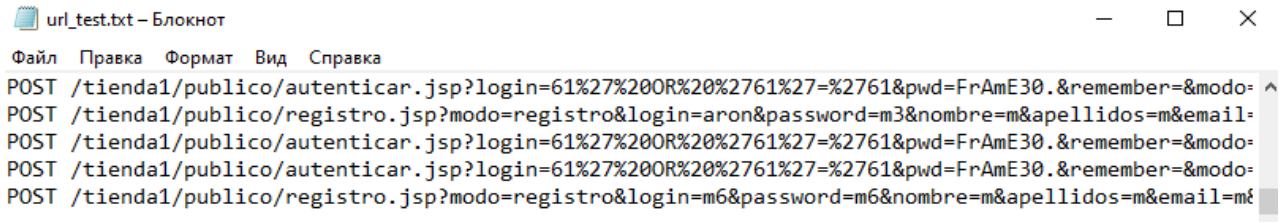


Рисунок 3.22 – Запити для верифікації моделі

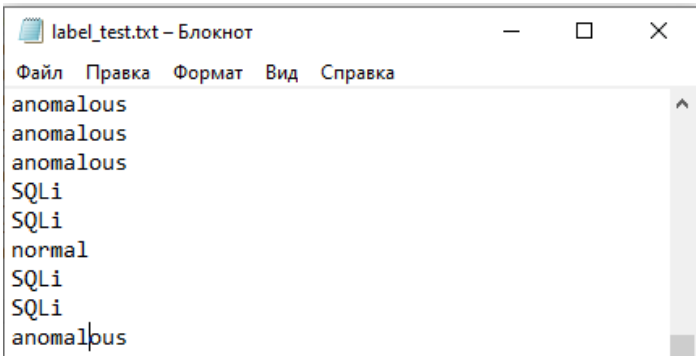


Рисунок 3.23 – Файл міток класів

В якості відображення класу UserInterface у діаграмі класів (рис. 1.4) був створений клас Detector (див. рис. 3.24). Програмна реалізація класу відповідно включає набір методів, щоб звертатись до окремої моделі формату h5 (cnn_clf.h5) та давати оцінку наявності зазначених кібератак одному або групі запитів.

За допомогою Detector, спеціаліст з ІБ може працювати з експертною системою. Цей клас є вхідною точкою у систему, стани якої послідовно змінюються відповідно до діаграми кінцевого автомату (див. рис. 1.5). Можна побачити, наприклад, що при створенні екземпляру Detector система перейде до стану ініціалізації, і тільки при її успішному завершені буде виконана обробка запитів.

```
class Detector:
    def __init__(self):
        self.tokenizer = pickle.load(open("./tokenizer/tokenizer.pickle", "rb"))
        self.model = load_model("./model/cnn_clf.h5")
        self.max_len = 600
        self.labels_type = ['SQLi', 'anomalous', 'normal', 'XSS', 'SSI', 'BufferO

    def props_to_labels(self, props_matrix):
        labels = []
        for props_vector in props_matrix:
            idx = np.argmax(props_vector)
            label = self.labels_type[idx]
            labels.append(label)
        return labels

    def predict_url(self, url):
        label_pred = self.predict_urls([url])[0]
        return label_pred

    def predict_urls(self, urls):
        seq = self.tokenizer.texts_to_sequences(urls)
        X = sequence.pad_sequences(seq, maxlen=self.max_len)
        Y_pred = self.model.predict(X)
        labels_pred = self.props_to_labels(Y_pred)
        return labels_pred
```

Рисунок 3.24 – Клас Detector

У стані «request processing» системи main-метод класу Detector (рис. 3.25) зчитує файл url_test.txt та, застосовуючи методи класу Detector надає передбачення першим 10 об’єктам UrlRequest щодо наявності зловмисного контенту.

```
if __name__ == '__main__':
    # for test
    detector = Detector()
    urls = []
    with open('./data/torpeda_train_test/url_test.txt', 'r', encoding='utf-8') as f:
        data = f.readlines()
        for d in data:
            urls.append(d[:-1])
    labels_pred = detector.predict_urls(urls[:10])
    print(labels_pred)
    label_pred = detector.predict_url(urls[0])
    print(label_pred)
```

Рисунок 3.25 – Main-метод класу Detector

Результат перевірки показує, що НМ стовідсотково вірно передбачила перші десять запитів, що показано на рисунку 3.26.

Рисунок 3.26 – Результат виконання програми

Таким чином, загальні результати дослідження свідчать про те, що:

1) Модель визначила частину трафіку Torpeda вразливою до кібератак.
2) Більш того, НМ ефективно впоралась з класифікацією кібератак у трафіку відповідно до визначений класів ін'єкцій: SQLi, XSS, SSI, BufferOverflow, CRLFi, XPath, LDAPi, FormatString.
3) Після аналізу роботи різних класифікаторів, зі зведеної таблиці метрик видно, що обрана згорткова модель є найбільш точною серед розглянутих.
4) Виявлення кожної кібератаки здійснюється з загальною ймовірністю 98% згортковою нейронною мережею. При цьому, ймовірність детектування SQLi складає 100%, XSS – 100%, BufferOverflow – 100%, CRLFi – 100%, LDAPi – 100%, SSI – 95%, XPath – 88%, FormatString – 73%.
5) Забезпечений також захист від запитів, клас яких неможливо визначити (які можуть бути атаками нульового дня) – їм притаманна позначка anomalous. Тестування показує, що точність виявлення аномальних запитів CNN - 99%.
6) Зазначений процент захищеності від кібератак у реальних умовах може бути зниженим через тимчасову непристосованість НМ до іншого трафіку.

3.3 Інтерфейс роботи з базою даних

cx_Oracle — це модуль розширення Python, який дозволяє надсилати запити й оновлювати бази даних Oracle за допомогою API бази даних, спільного для всіх програмних модулів доступу до БД.

На початку визначимо, що у середі Oracle існує БД orcl з таблицями HTTP_Request, HTTP_Type, Model_Request, NeuroModel. Дані таблиці створив користувач БД security1, тобто він є їх власником та має усі привілеї на їх використання. На рисунку 3.27 представлений набір таблиць, який можна побачити у програмі PL/SQL Developer (це інтегроване середовище розробки, яке призначене для розробки процедур для БД Oracle).

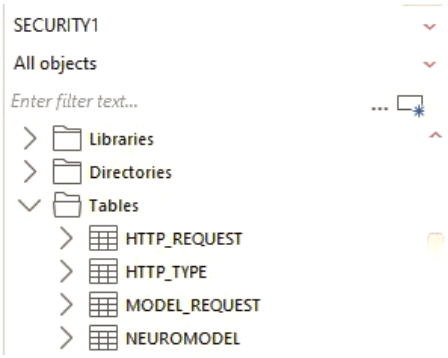


Рисунок 3.27 – Таблиці користувача security1

Щоб підключитися до бази даних orcl, потрібно імпортувати модуль cx_Oracle і далі викликати метод два методи:

– makedsn() - використовується для створення рядка дескриптора підключення, функція приймає ім’я хоста бази даних (у даному випадку DESKTOP-EEOFLDP), номер порту (1521) та назву служби (orcl);
– connect(), у якому передані параметри визначають користувача БД, його пароль для авторизації та дескриптор підключення (див. рис. 3.28). Метод cursor() відкриває курсор для використання операторів. Метод execute() аналізує та виконує оператор, переданий як строковий параметр.

```
print(cx_Oracle.version)
try:
    dsn_tns = cx_Oracle.makedsn('DESKTOP-EEOFLDP', '1521', service_name='orcl')
    conn = cx_Oracle.connect(user=r'security1', password='q1w2e3r4t5', dsn=dsn_tns)
except cx_Oracle.Error as error:
    print(error)
c = conn.cursor()

c.execute('select * from HTTP_TYPE where type_id = 6')
res = c.fetchall()
for row in res:
    print(row)
```

```
8.3.0
(6, 'BufferOverflow')
```

Рисунок 3.28 – Підключення до бази даних Oracle

Підключення є встановленим, адже параметри функцій вибирались у відповідності до файлу ~/ORACLE_HOME/network/admin/tnsnames.ora, це файл конфігурації SQL*Net, який визначає адреси баз даних для встановлення з'єднань з ними. Він представлений на рисунку 3.29.

tnsnames.ora

```
19  ORCL =
20    (DESCRIPTION =
21      (ADDRESS = (PROTOCOL = TCP)(HOST = DESKTOP-EEOFLDP)(PORT = 1521))
22      (CONNECT_DATA =
23        (SERVER = DEDICATED)
24        (SERVICE_NAME = orcl)
25      )
26    )
```

Рисунок 3.29 – Файл конфігурації tnsnames.ora

Перевірка після підключення показує, що з бази даних orcl, таблиці HTTP_Type був отриманий рядок даних, який відповідає дійсності, тобто підключення встановлене та стабільне (див. рис. 3.28).

Після встановлення підключення та навчання НМ можна також зберегти робочу модель з усіма здобутими у результаті навчання параметрами вагових коефіцієнтів і зв'язками, і з метриками точності. На рисунку 3.30 зображений варіант збереження раніше навчених моделей у БД.

Перевірка коректності виконання даного коду представлена на рисунку 3.31, у якому проводиться звернення до таблиці Моделей та відбувається вибірка усіх записів. Можна побачити, що збереглись усі моделі.

```
def upload_model_to_db(num, file, date, acc, comment):
    sql = ('insert into NeuroModel(model_id, model_file, model_date, model_accurancy, model_comment) '
        'values(:num, :file, :date, :acc, :comment)')
    c.execute(sql, [num, file, date, acc, comment])
    conn.commit()

nns = { "CNN": model, "DecisionTree": DT, "KNN": knn_clf, "RandomForest": rf_clf }
i = 0

for key in nns:
    file_path = "./model/" + key + ".h5"
    model = nns[key]
    model.save(file_path)
    upload_model_to_db(i, file_path, date.today(), accuracy_dict[key], "")
    i += 1
```

Рисунок 3.30 – Зберігання моделей у БД

```
c.execute('select * from NEUROMODEL')
res = c.fetchall()
for row in res:
    print(row)
```

```
(0, <cx_Oracle.LOB object at 0x0000023584B727E0>, datetime.datetime(2022, 11, 2, 0, 0), 0.995638, None)
(1, <cx_Oracle.LOB object at 0x0000023584B47930>, datetime.datetime(2022, 11, 2, 0, 0), 0.983408, None)
(2, <cx_Oracle.LOB object at 0x0000023584B47420>, datetime.datetime(2022, 11, 2, 0, 0), 0.973651, None)
(3, <cx_Oracle.LOB object at 0x00000235873BCFC0>, datetime.datetime(2022, 11, 2, 0, 0), 0.983049, None)
```

Рисунок 3.31 – Перевірка завантаження моделей до БД

Відповідно до спроектованих у першому розділі UML діаграм був створений клас Database, у якому визначений основний API взаємодії з конкретними таблицями бази даних. На основі діаграми класів (рис. 1.4), реалізовані наступні методи: підключення до БД, яке встановлюється при створенні об'єкту бази даних (тобто у конструкторі класу), метод оновлювання таблиці HTTP_Request новими запитами та таблиці Model_Request новими зв'язками Запит-Модель (див. рис. 3.32), а також методи, які були розглянуті у файлі навчання моделей – завантаження моделей до БД, вибірка моделей, вибірка типів запитів.

Метод завантаження запитів використовується після того, як модель зробила передбачення атак. У цьому випадку до БД будуть завантажені атаки на веб-сервіс. Існує також проміжний етап, коли після прогнозування атак спеціаліст знайшов помилку у висновках моделі. Тоді спеціаліст корегує прогнозування та метод на рис. 3.32 зберігає до БД не тільки атаки, але й помилки. Для визначення запиту як атаки або помилки використовується атрибут attack_error, де значення 0 – кібератака, а 1 – помилка моделі.

```
27      def upload_requests(self, labels, x, y, model, attack_error):
28          http_request = ('insert into HTTP_REQUEST(http_id, http_query, http_date, http_type_id, http_danger_lvl, '
29                          'attack_error) values(:http_id, :query, :date, :type, :lvl, :attack_error)')
30          model_request = ('insert into MODEL_REQUEST(model_http_id, model_id, http_id) '
31                           'values(:id, :model_id, :http_id)')
32
33          ht_id = get_http_id() + 1
34          mod_req_id = get_model_request_id() + 1
35          model_id = self.get_model_id(model)
36
37          for i in [0..len(x)]:
38              if !(y[i] != 'normal' && attack_error[i] == 0):
39                  self.c.execute(http_request, [ht_id, x[i], date.today(), labels.index(y[i]),
40                      accuracy[i], attack_error[i]])
41                  self.c.execute(model_request, [mod_req_id, model_id, ht_id])
42                  ht_id += 1
43                  mod_req_id += 1
44
45          self.conn.commit()
```

Рисунок 3.32 – Зберігання запитів у БД

Серед додаткових методів класу Database були реалізовані наступні. Метод для отримання id нейронної моделі, метод для отримання id останнього запису таблиці HTTP_Request, щоб продовжити додавання нових записів, а не переписувати усю таблицю, метод для отримання id останнього запису таблиці Model_Request, щоб також продовжити додавати нові рядки з місця останнього доданого. Програмна реалізація методів наведена на рисунку 3.33.

```
45      def get_model_id(self, model):
46          self.c.execute('select model_id from NEUROMODEL where model_file = :model_file', [model.file_path])
47          res = self.c.fetchall()
48          return res
49
50      def get_http_id(self):
51          self.c.execute('SELECT MAX(http_id) FROM HTTP_REQUEST')
52          res = self.c.fetchall()
53          return res
54
55      def get_model_request_id(self):
56          self.c.execute('SELECT MAX(model_http_id) FROM MODEL_REQUEST')
57          res = self.c.fetchall()
58          return res
```

Рисунок 3.33 – Зберігання запитів у БД

Перевірка коректності збереження запитів представлена на рисунку 3.34. Оскільки у класі не реалізована вибірка запитів з таблиці HTTP_Request, перевірка відбувалася у PL\SQL Developer.

З перших десяти рядків можна побачити, що відбулось завантаження запитів з відображенням id типу цього запита, при цьому, серед запитів немає типу normal, адже відбувалось завантаження тільки атак. Але у ситуації, коли спеціаліст з ІБ знайде помилку у передбаченні моделі для запиту, який фактично не є normal, запит буде збережений з типом normal та атрибутом attack_error = 0.

	HTTP_ID	HTTP_QUERY	HTTP_DATE	HTTP_TYPE_ID	HTTP_DANGER_LVL	ATTACK_ERROR
1	1	GET /tienda1/publico/carac ···	05.11.2022 ···	1	0,88	1
2	2	POST /tienda1/publico/regi ···	05.11.2022 ···	4	0,89	1
3	3	0%2C98%2C114%2C62%2C ···	05.11.2022 ···	1	0,95	1
4	4	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,93	1
5	5	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,93	1
6	6	POST /tienda1/publico/aut ···	05.11.2022 ···	1	0,91	1
7	7	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,96	1
8	8	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,95	1
9	9	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,89	1
10	10	POST /tienda1/publico/aute ···	05.11.2022 ···	1	0,92	1

Рисунок 3.34 – Перевірка завантаження запитів до БД

Метод отримання нейронної моделі, який використовується на етапі ініціалізації програми для завантаження моделі перед її прогнозуванням кібератак (див. рис. 3.35). За замовчування у інтерфейсі користувача обирається модель з найвищим показником точності.

```
64      def load_model(self):
65          self.c.execute('SELECT MAX(model_accurancy) FROM NEUROMODEL')
66          accuracy = self.c.fetchall()
67          self.c.execute('select model_file from NEUROMODEL where model_accurancy = :accuracy', [accuracy])
68          res = self.c.fetchall()
69          return res
```

Рисунок 3.35 – Завантаження моделі нейронної мережі

Також згідно діаграм кінцевого автомату (рис. 1.5), перед тим як вийти з програмного інтерфейсу, система повинна перейти до стану «closing program». Програма виконує метод clean(), тобто завершити з'єднання з БД та подальше використання курсору для виконання SQL-команд. Програмна реалізація методу представлена на рисунку 3.36.

```
60      def clean(self):
61          self.c.close()
62          self.conn.close()
```

Рисунок 3.36 – Завершення сесії з БД

Синхронізована робота різних елементів системи відстежується за допомогою діаграми діяльності (рис. 1.6). Detector User Interface (клас Detector) координує роботу усіх інших елементів після запитів спеціалісту з ІБ до системи.

Отже, за наявністю реалізації та перевірки функціонування вищенаведених методів комунікації програмного інтерфейсу та БД, а також за наявністю обраної найкращим чином моделі нейронної мережі, яка пройшла успішне навчання і верифікацію, вважається, що мета розробки повноцінної нейронної експертної системи досягнута.

3.4 Рекомендації зі застосування експертної системи

Загальні рекомендації забезпечення безпеки в ІТС враховують не тільки класичні механізми і заходи, але і включають специфічні методи для конкретного домену [5]. На сьогоднішній день, вимоги до якісного засобу захисту зростають

пропорційно швидкості як розвитку нових методів кібератак, так і загалом розвитку IT-індустрії, що потребує постійного вдосконалення концепції захисту.

Визначимо пункти, які повинні бути здійснені для забезпечення грамотного та дійсно професійного захисту конкретного веб-ресурсу:

– Аналіз ризиків ІБ і, як наслідок, визначення пріоритетів та ступеню захисту інформації на їх основі.
– Вибір сприятливих методів і засобів захисту, які разом забезпечують повне перекриття від загроз. Іноді існуючих пропозицій на ринку недостатньо для забезпечення даного етапу, тому й відбувається розробка власних засобів.
– Гармонійне налаштування і поєднання загальновідомих підходів забезпечення захищеності з провідними, які відповідають стрімкому розвитку нових способів проведення атак на підприємства.

Звісно, один окремий засіб малоймовірно буде здатний забезпечити повне перекриття від загроз. Більш того, питання кібербезпеки має бути враховано на кожному рівні веб-ресурсу, від проектування до підтримки. Тому рекомендується застосовувати комплексний підхід. Неможливо одразу врахувати усі можливі фактори безпеки, але деякі важливі рекомендації наведені нижче:

– Експлуатації сучасних технічних засобів захисту:
    - розглянути можливість побудови веб-ресурсу на основі мікросервісів (архітектура MACH), що дозволить уникнути відмови усієї системи у разі кібератаки і відокремити залежності між окремими сервісами;
    - встановлення надійної системи управління оновленнями ПЗ;
    - встановлення надійної системи антивірусного захисту;
    - використання SIEM (англ. Security information and event management) – управління кіберінцидентами у реальному часі з метою зменшити час реагування та збитки;
    - ефективний віддалений доступ. Хмарні технології стають потужною і багатообіцяючою перевагою для дистанційного керування безпекою;
    - розглядання можливості застосування міжмережевих екранів.

– Захист конфіденційних даних:
  - зберігання ІзОД у недоступному (зашифрованому) вигляді та/або підтримка обмеженого доступу до неї;
  - регулярне створювання резервних копій систем і зберігання їх на виділених серверах окремо від робочої мережі;
  - мінімізація привілеїв користувачів і служб;
  - використовування різних облікових записів і паролів доступу до різних ресурсів;
  - розумне забезпечення багатофакторної автентифікації (наприклад, для захисту привілейованих облікових записів).
– Обов'язкове встановлення складних паролів:
  - підтримання парольної політики, яка передбачає суворі вимоги до мінімальної довжини, алфавіту і складності паролів;
  - регулярне оновлення паролів (не більше 90 днів).
– Постійний моніторинг і покращення безпеки системи з плином часу:
  - підвищення обізнаності нових робітників і клієнтів про важливість ІБ;
  - регулярне проведення тестування на проникнення;
  - фільтрація трафіку для мінімізації доступних зовнішньому зловмисникові інтерфейсів мережевих служб;
  - регулярне проведення аналізу захищеності веб-додатків, включаючи аналіз вихідного коду, з метою виявлення та усунення вразливостей, що дозволяють проводити атаки, в тому числі на клієнтів додатки;
  - контроль появи небезпечних ресурсів на периметрі мережі;
  - регулярне проведення інвентаризації ресурсів, доступних для підключення з інтернету, аналіз захищеності таких ресурсів і усунення вразливості в використовуваному ПЗ.

Щодо рекомендацій зі застосування експертної системи, можна сказати, що спеціалісту з ІБ пропонується запустити інтерфейс користувача (Detector), ввести файл запитів у форматі txt, проаналізувати надані результати і виправити їх (алгоритм виправлення наведений на рис. 3.37), далі – оновити БД новими даними.

Дана частина програми реалізована на основі діаграм кінцевого автомату (стани після етапу «request processing») та діяльності системи (сутність Security Specialist виконує «Confirm prediction» або «Correct prediction»).

detector.py

```
99      # confirm prediction
100     edit = (len(labels_pred))
101     np.zeros(edit)                    #0 - attack, 1 - error
102
103     print('Do you want to correct model prediction? 1 - yes, 0 - no')
104     if (input() == 1):
105         print('Correct each HTTP Request (0 - attack, 1 - error): ')
106         for i in [0..len(labels_pred)]:
107             print(" %-*s %s" % (13, labels_pred[i], urls[i]))
108             edit[i] = input()
109
110     db = Database()
111     db.upload_requests(urls, Detector.labels_pred, detector.model)
112     db.clean()
```

Рисунок 3.37 – Алгоритм виправлення результатів НМ

Додатковим джерелом знань про те, як працює нейронна експертна система, є візуалізація за допомогою UML-діаграм (рис. 1.5, 1.6, 1.7). Їх можна використовувати у якості частини документації та опису системи з метою надання новим користувачам можливості швидко ознайомитись з її компонентами та процесом їх взаємодії. Наприклад, за допомогою діаграми діяльності (рис. 1.6), спеціаліст з ІБ може побачити необхідні з його боку кроки роботи з системою.

Не рекомендується застосовувати експертну нейронну систему як єдине джерело забезпечення кіберзахисту, вона вимагає підтримки і постійного контролю, налаштування і верифікації обізнаних спеціалістів з ІБ.

За результатами виконання роботи, можна сказати, що розроблена нейронна експертна система повністю відповідає очікуваним результатам та виставленим вимогам, усі її елементи розумно та обґрунтовано обрані, спроектовані та успішно поєднані. Окрім цього, протестовані основні випадки використання. Отже, мета даної роботи вважається досягнутою.

ВИСНОВКИ

1) У роботі обґрунтовано рішення про заміщення бази знань експертної системи нейронною мережею на підставі аналізу релевантних наукових публікацій та досліджень. Сформовані вимоги до функціонування експертної системи і проілюстровані відповідні UML-діаграми (класів, кінцевого автомату, діяльності).

2) Проведено аналіз релевантних загроз ІБ за рейтингами CWE та OWASP, на основі якого був визначений список ін'єкцій для детектування: SQL Injection, Cross Site Scripting, XML External Entities Injection, Server-Side Includes Injection, Buffer Overflow, Carriage Return Line Feed Injection, XPath, Lightweight Directory Access Protocol Injection, Format String.

3) Розроблена база даних нейромережевого детектору зловмисного трафіку до веб-серверу та механізми її захисту. Проектування БД здійснювалось за допомогою концептуального, логічного та фізичного моделювання.

4) Розроблені механізми захисту БД, а саме: розмежування доступу здійснюється на основі призначених привілей (системних, об'єктних) та ролей призначених кожному користувачу відповідно до його рівня кваліфікації, розроблений профіль, якій контролює ресурси БД та процес входу до облікового запису. Усі застосовані механізми захисту були перевірені на справність.

5) Дослідження нейронної мережі показали, що архітектура CNN є найбільш точною і ефективною у порівнянні з K-NN, Decision Trees, Random Forest, адже вона надає 99% загальної точності класифікації кібератак. Також, помилки першого та другого родів у відповідності до використаного набору даних навчання і тестування Torpeda складають менше 5%.

6) Усі елементи розробленої системи були поєднані за допомогою механізму з'єднання з базою даних та програмного інтерфейсу користувача. Показані основні випадки застосування БД, а також програмного коду виправлення помилок моделі для подальшого перенавчання і покращення точності аналізу реального трафіку.

7) Сформовані як загальні рекомендації для забезпечення безпеки ІТС, так і поради застосування нейронної експертної системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рогоза П., Єсін, В. Використання нейронної мережі замість бази знань у експертній системі детектору зловмисного трафіку до веб-ресурсів – 2022, Комп'ютерні науки та кібербезпека, (1), 4-14. URL: https://doi.org/10.26565/2519-2310-2022-1-01 (дата звернення: 22.08.2022).
2. Liao S. H. Expert system methodologies and applications—a decade review from 1995 to 2004. Expert systems with applications. – 2005. URL: https://doi.org/10.1016/j.eswa.2004.08.003 (дата звернення: 02.09.2022).
3. Tan H. A brief history and technical review of the expert system research //IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2017. URL: https://doi.org/10.1088/1757-899X/242/1/012111 (дата звернення: 02.09.2022).
4. Hoffman L. J., Modern Methods for Computer Security and Privacy: John Wiley & Sons, 1973. 264 с.
5. Рогоза П. В. Оцінка захисту web-ресурсів від кібератак. Пояснювальна записка до дипломної роботи бакалавра: напрям підготовки 125 – Кібербезпека / П. В. Рогоза; Харківський національний університет імені В. Н. Каразіна. – Харків: [Б. В.], 2021. – 67 с.
6. Корченко О., Терейковський І., Дзюбаненко А. Сучасні нейромережеві методи та моделі оцінки параметрів безпеки ресурсів інформаційних систем: Захист Інформації, том 16, №3, липень-вересень 2014.
7. Batista L. O. et al. Fuzzy neural networks to create an expert system for detecting attacks by sql injection. 2019. URL: https://doi.org/10.48550/arXiv.1901.02868 (дата звернення: 04.09.2022).
8. Mahdavifar S., Ghorbani A. A. DeNNeS: deep embedded neural network expert system for detecting cyber-attacks. Neural Computing and Applications. 2020. URL: https://doi.org/10.1007/s00521-020-04830-w (дата звернення: 04.09.2022).

9. OWASP Top 10 Application Security Risks. 2021. URL: https://owasp.org/Top10/ (дата звернення: 09.09.2022).
10. Common Weakness Enumeration (CWE) Top 25 Most Dangerous Software Weaknesses. 2022. https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html (дата звернення: 09.09.2022).
11. Єсін В. І., Кузнецов О. О., Сорока Л. С. Безпека інформаційних систем і технологій. Х.: ХНУ імені В. Н. Каразіна, 2013. 632 с.
12. Connolly T. M., Begg C. E. Database systems: a practical approach to design, implementation, and management. Sixth edition. Harlow, Essex, England: Pearson Education Limited, 2015. 1329 p.
13. Gaetjen S., Knox D., Maroulis W. Oracle Database 12c Security. McGraw-Hill Education, 2015. 526 p.
14. Oracle Database SQL Language Reference 19c. E96310-11 – August 2021. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/index.html (дата звернення: 12.09.22).
15. Sibi P., Jones S. A., Siddarth P. Analysis of different activation functions using back propagation neural networks. Journal of theoretical and applied information technology. 2013. 47(3). P. 1264-1268. (дата звернення: 05.10.22).
16. Keras. URL: URL: https://ru-keras.com/guide-sequential/ (дата звернення: 12.10.22).
17. Дунець Р. Б., Рак Ю. П., Зачко О. Б. Класифікація територій засобами нейронних мереж для управління проектами в забезпеченні екологічної безпеки. 2008. URL: https://sci.ldubgd.edu.ua/bitstream/123456789/2505/1/08_23.pdf (дата звернення: 13.10.22).
18. Torrano C., Perez A., Alvarez G., What is Torpeda. 2012. URL: https://www.tic.itefi.csic.es/torpeda/default.html (дата звернення: 17.10.22).
19. Соснін А. С., Суслова И. А., Функции активации нейросети: сигмоида, линейная, ступенчатая, RELU, TAHN. Русский государственный профессионально-педагогический университет, Екатеринбург. 2019.
20. Christopher M., Pattern Recognition and Machine Learning. Springer, 2006.

21. Lewandowski M., Makris D., Nebel J.-C., View and Style-Independent Action Manifolds for Human Activity Recognition. European Conference on Computer Vision, ECCV 2010: Computer Vision, ECCV 2010. P. 547-560. URL: 10.1007/978-3-642-15567-3_40 (дата звернення: 01.11.22).
22. Janocha K., Czarnecki W. M. On loss functions for deep neural networks in classification. arXiv preprint arXiv:1702.05659. 2017. URL: https://arxiv.org/abs/1702.05659 (дата звернення: 01.11.22).
23. Kingma D. P., Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014. URL: https://arxiv.org/abs/1412.6980 (дата звернення: 03.11.22).

ДОДАТОК А

СПИСОК ПУБЛІКАЦІЙ МАГІСТРА

ISSN 2519-2310 CS&CS, Issue 1(21) 2022

УДК 681.04

ВИКОРИСТАННЯ НЕЙРОННОЇ МЕРЕЖІ ЗАМІСТЬ БАЗИ ЗНАНЬ У ЕКСПЕРТНІЙ СИСТЕМІ ДЕТЕКТОРУ ЗЛОВМИСНОГО ТРАФІКУ ДО ВЕБ-РЕСУРСІВ

Поліна Рогоза, Віталій Єсін

Харківський національний університет імені В.Н. Каразіна, майдан Свободи, 4, Харків, 61022, Україна
polina.rohoza@gmail.com, v.i.yesin@karazin.ua

Рецензент: В'ячеслав Калашников, д. ф.-м.н., проф., Технологічний університет Монтеррея, 64849 Монтеррей, Нуево-Леон, Мексика
kalash@itesm.mx

Надійшло: Червень 2022.

Анотація: *Сучасний світ інформаційних технологій надає нам широкий спектр веб-застосунків. Звісно, існує постійна необхідність у надійному захисті веб-ресурсів та конфіденційної інформації, яка на них зберігається. Зі збільшенням числа кібератак зростають також критичні наслідки від них не тільки для приватних осіб, але і для підприємств. В роботі розглянуто елементи експертної системи та здійснено оцінювання їх ефективності. Основна мета застосування експертної системи – підвищення захищеності веб-ресурсів від кібератак (типу SQLi, XSS, SSI, BufferOverflow тощо) шляхом забезпечення швидкої обізнаності фахівців інформаційної безпеки про наявність атаки. Нейронна мережа здатна детектувати та класифікувати зловмисний трафік веб-серверів. До переваг застосування нейронної мережі відносяться: ефективна побудова нелінійних залежностей, адаптація до змін та оцінювання атак "нульового дня", відмовостійкість, відносна простота реалізації, швидкість обчислення після навчання. Результатом роботи є розроблений елемент експертної системи – навчена та верифікована модель нейронної мережі, яка гарантує 98% успішності детектування кібератак на веб-ресурси, а також менше 5% виникнення помилок першого та другого роду у відповідності до використаного набору даних.*

Ключові слова: *експертна система; нейронна мережа; захист веб-додатків; кібератака.*

1. Вступ

Підвищення захищеності веб-ресурсів є однією з найважливіших сфер інформаційної безпеки (ІБ). Збільшення кількості веб-ресурсів неминуче веде за собою до збільшення числа кібератак, погіршуються наслідки від цих атак них не тільки для приватних осіб, але і для підприємств. Проблема постійного збільшення чисельності кібератак потребує своєчасного інформування фахівців ІБ про поточний стан кіберзагроз. На сучасному етапі технологічного розвитку є кілька підходів для вирішення зазначеної проблеми, зокрема за рахунок впровадження автоматичних інтелектуальних систем. Однак, нині засоби захисту, які засновані на штучних нейронних мережах, досі надають широкий потенціал для різних наукових випробувань, зокрема для захисту веб-ресурсів, і не є повноцінно дослідженими. Враховуючи те, що застосування технології нейронних мереж (НМ) надає багато практичних можливостей і переваг, безумовно, дана тематика є вкрай актуальною.

В межах вирішення зазначеної проблематики в даній роботі розглядається можливість використання експертної системи (ЕС), що побудована на основі технологій НМ. Дана ЕС дозволяє детектувати зловмисний трафік веб-серверів. Створення автономної ЕС дозволяє знизити навантаження на фахівців ІБ, підвищити швидкість аналізу трафіку та виключити «людський фактор». В цілому це дозволить зменшити втрати підприємств (економічні, інформаційні, репутаційні) та підвищити рівень захищеності веб-застосунків від кібератак.

2. Основні поняття та пов'язані роботи

Експертна система – це комп'ютерна система штучного інтелекту, призначена для вирішення складних проблем (*прогнозування, контролювання, управління тощо*) і надання мо-

