Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів глибинного машинного навчання для вирішення задачі побудови системи виявлення вторгнень
(тема)

Виконав: студент 2 курсу, групи ПЗСм-19-1
спеціальності 121-Інженерія програмного забезпечення
(код і повна назва спеціальності)
Освітньо-професійної програми
Програмне забезпечення систем
(повна назва освітньої програми)
Попович І.Д.
(прізвище, ініціали)
Керівник доц. Мазурова О.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В. Дудар
(підпис)

2020 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий(магістерський)

Спеціальність 121-Інженерія програмного забезпечення
(код і повна назва)

Освітньо-професійна програма Програмне забезпечення систем
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«_____»_____ 2020 р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Поповичу Івану Дмитровичу*

1. Тема роботи Дослідження методів глибинного машинного навчання для вирішення задачі побудови системи виявлення вторгнень

затверджена наказом по університету від «30 » 10 2020 р. № 1490 Ст

2. Термін подання студентом роботи (проекту) 10.12.2020

3. Вихідні дані до роботи (проекту) *модель глибинного машинного навчання для вирішення задачі будування системи виявлення вторгнень, система моніторингу мережевого трафіку, пояснювальна записка*.

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити) *вступ, аналіз проблемної області і постановка задачі, перелік вимог до програмної системи, опис дослідження, об'єктних моделей, дослідження методів та алгоритмів, опис розробленої програмної реалізації, аналіз можливих застосувань та експлуатації системи.*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів) *постановка задачі, об'єктна модель системи, базові моделі, архітектура додатку, структура бази даних, приклади інтерфейсів програмної системи, висновки, слайди презентації.*

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доц. Мазурова О.О.		

КАЛЕНДАРНИЙ ПЛАН

	Назва етапів роботи	Термін виконання етапів	Примітка
1	Аналіз проблемної області	1.09.20–30.09.20	виконано
2	Аналіз існуючих методів	30.09.20–10.10.20	виконано
3	Аналіз та моделювання предметної області	10.10.20–12.10.20	виконано
4	Аналіз методів машинного навчання	12.10.20–14.10.20	виконано
5	Підготовка даних	14.10.20–16.10.20	виконано
6	Тестування моделей	16.10.20–19.10.20	виконано
7	Проектування схеми БД	19.10.20–25.10.20	виконано
8	Проектування архітектури системи	25.10.20–10.11.20	виконано
9	Створення коду програми	11.11.2020–28.11.2020	виконано
10	Тестування програми	28.11.2020–30.11.2020	виконано
11	Підготовка пояснювальної записки.	1.11.2020–19.11.2020	виконано
12	Підготовка презентації та доповіді	20.11.2020–9.12.2020	виконано
13	Попередній захист	10.12.20	виконано
14	Нормоконтроль, рецензування	14.12.20	виконано
15	Занесення диплома в електронний архів	14.12.20	виконано
16	Допуск до захисту у зав. кафедри	15.12.20	

Дата видачі завдання 1.09.2020

Студент _____ Попович І.Д.

(підпис)

Керівник роботи _____ доц. Мазурова О.О.

(підпис)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 93 с., 27 рис., 3 табл., 26 дж.

ІНФОРМАЦІЙНА БЕЗПЕКА, МАШИННЕ НАВЧАННЯ, НЕЙРОННА МЕРЕЖА, ТРАФІК, LINUX, POSTGRESQL, PYTHON, UML.

Об’єктом дослідження є методи глибинного машинного навчання в контексті побудови мережевої системи виявлення вторгнень.

Метою роботи є дослідження методів глибинного машинного навчання для розробки програмної системи мережевого виявлення вторгнень, яку можна використати для аналізу трафіку та сповіщення користувачів про загрози.

Методи розробки програмної системи базуються на використанні операційної системи Linux, мов програмування Python та JavaScript, системи управління базами даних PostgreSQL, сервісу RabbitMQ.

У результаті роботи проведено дослідження методів глибинного машинного навчання, спроектована схема бази даних та архітектура додатку та розроблена система мережевого виявлення вторгнень.

INFORMATION SECURITY, MACHINE LEARNING, NEURAL NETWORK, LINUX, POSTGRESQL, PYTHON, TRAFFIC, UML.

The object of the study is deep learning methods that can be used as base model for building network intrusion detection system.

The purpose of the work is to analyse deep learning methods for designing a software system for network intrusion detection based on deep learinng methods.

The software development methods are based on Linux operating system, Python and JavaScript programming languages, PostgreSQL database management system and RabbitMQ message broker.

As a result, deep learing model is trained that can be used in different monitoring tools and network analyzers. Network intrusion detection system was developed based on the conducted research.

ЗМІСТ

ВСТУП

Інтернет технології з кожним роком стають все більш важливою частиною повсякденного життя людини. Тому актуальність проблеми забезпечення інформаційної безпеки збільшується швидкими темпами. З розвитком технологій з'являються нові можливості та інструменти для проведення кібератак, що в свою чергу активізує процес розробки нових засобів захисту та навпаки [1].

Сучасні системи складаються з клієнтських компонентів, з якими взаємодіють користувачі, і серверних компонентів, які надають функції для клієнтів.

Незважаючи на те, що серверні компоненти не мають безпосереднього впливу на кінцевих користувачів, вони є найбільш вразливими елементами системи [2], оскільки змушені не тільки відправляти, а й приймати дані відразу декількох клієнтів. Тому забезпечення безпеки серверів є ключовим елементом в контексті безпеки системи.

Взаємодії через мережу – основний спосіб впливу на сервери для зловмисника, тобто швидке виявлення зловмисних мережевих активностей – важливий етап для забезпечення безпеки інтернет ресурсів [2]. Саме цю проблему вирішують системи виявлення вторгнень (англ. NIDS) – додатки, що аналізують мережевий трафік з метою виявлення підозрілих елементів [4].

Формально NIDS виконує функцію класифікації трафіку на шкідливий і звичайний [5]. Зараз завдання класифікації досить ефективно вирішують за допомогою методів глибинного машинного навчання, які активно розвиваються останні кілька років.

Більшість сучасних програмних систем у якості серверу використовують хмарні рішення. Найбільш поширені сервіси, які надають великі провайдери, такі як AWS, Google Cloud, Azure тощо [3]. Вони є надійним рішенням в тому числі з точки зору безпеки і мають велику кількість функцій. Головний їх недолік – ціна. Також вони є громіздкими з точки зору досвіду користувача і вимагають великих

затрат часу на навчання. З огляду на ці недоліки багато бізнесів обирають більш малі хмарні провайдери, такі як DigitalOcean, Vultr або Linode. Вони потребують менше грошових витрат, але мають набагато менше функціоналу, зокрема, забезпечення безпеки лягає на плечі користувача системи.

Метою роботи є дослідження методів глибинного машинного навчання в контексті вирішення завдання розробки системи NIDS, заснованої на методах глибинного машинного навчання, яку можна буде використати для аналізу шкідливого мережевого трафіку у кластері, вузли якого розгорнуті на будь-якому з провайдерів хмарних обчислень.

Крім того, система повинна бути обладнана системами моніторингу, нотифікації зловмисних активностей, надавати можливість відображати статистику оброблених мережевих даних для подальшого аналізу, а також бути досить легкою у використанні та інсталяції.

Теоретичною основою роботи є праці вітчизняних і зарубіжних вчених, в яких розглядаються практичні і теоретичні питання в області інформаційної безпеки, архітектури програмних систем та мереж, машинного навчання.

В ході роботи застосовувалися методи системного аналізу, порівняння, узагальнення, моделювання. Отримані результати, модель на основі глибинного навчання та програмна система мережевого виявлення вторгнень, можуть бути застосовані адміністраторами та інженерами програмних систем мереж у якості базового інструмента моніторингу загрозливого трафіку в хмарних кластерах.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток Б). Крім того, зроблено доповідь на Міжнародному молодіжному форумі. Тези доповіді наведено в додатку В. Також за результатами роботи були опубліковані тези та зроблена доповідь на конференції «Всеукраїнська науково-практична конференція. Комп’ютерна інженерія і кібербезпека: досягнення та інновації» 25-27 листопада 2020 року (див. додаток Г).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області системи виявлення вторгнень

Інформаційні системи давно є безпосереднім елементом сучасного життя. Через менші витрати на встановлення та обслуговування багато з цих систем в значній мірі взаємодіють між собою за допомогою мережі [1]. Збереження конфіденційності, безпеки даних та запобігання несанкціонованому доступу є ключовими завданнями, які вирішуються для забезпечення функціонування систем [2].

Сервери як елементи інформаційної системи є найбільш вразливими до кібератак компонентами, бо вони мають відкриті канали як прямої, так і зворотної комунікації та при цьому обробляють велику кількість чутливої інформації [6]. Тому виявлення потенційних загроз є одним із важливих заходів для забезпечення безпеки .

Кібератаки поділяють на два типи: активні та пасивні [7]. Мета перших – вивести з ладу системні ресурси або вплинути на їх роботу. Пасивні, в свою чергу, намагаються дізнатись або використати інформацію із системи, не впливаючи на її працездатність. В сучасних методах вторгнення пасивні атаки застосовуються як продовження активних [4]. Тому важливо мати можливість якнайшвидше реагувати на активні атаки.

До основних типів активних кібератак належать наступні загрози [3]:

– DoS (англ. Denial of Service);
– внутрішня атака;
– інжекція структурованої мови запитів;
– інжекція команд;
– атаки грубої сили.

DoS (англ. Denial of Service) – хакерська атака на обчислювальну систему з метою довести її до відмови, тобто створення таких умов, при яких сумлінні користувачі системи не зможуть отримати доступ до системних ресурсів, або цей

доступ буде ускладнено [8]. Також такий тип атаки може бути і кроком до оволодіння системою (якщо в нештатній ситуації ПО видає будь-яку критичну інформацію – наприклад, версію, частину програмного коду тощо). Але частіше це міра економічного тиску: система, яка не може функціонувати деякий час, може принести значні збитки. В даний час DoS-атаки найбільш популярні у зв'язку з тим, що дозволяють довести до відмови практично будь-яку систему, не залишаючи юридично значимих доказів [3].

Внутрішня атака – кібератака, яка полягає в отриманні доступу до одного з вузлів за допомогою методів соціальної інженерії (розсилки спаму на email, встановлення шкідливого ПО на комп'ютери користувачів тощо) з метою подальших заходів з компрометації системи [5].

Інжекція структурованої мови запитів (англ. SQL injection) – це техніка введення коду, яка використовується для модифікації або отримання даних із баз даних SQL [9]. Вставляючи спеціалізовані оператори SQL у поле введення, зловмисник може виконувати команди, які дозволяють отримувати інформацію з бази даних, знищувати конфіденційні дані, або реалізовувати інші засоби маніпулятивної поведінки.

Інжекція команд (англ. Command injection) – атака, метою якої є виконання довільних команд на головній операційній системі через вразливий додаток [10]. Атаки введення команд можливі, коли програма передає небезпечні дані, надані користувачем (форми, файли cookie, заголовки HTTP тощо) в системну оболонку. У цій атаці команди операційної системи, що надаються зловмисником, зазвичай виконуються з привілеями вразливої програми [11]. Атаки введення команд можливі здебільшого через недостатню перевірку та валідацію вхідних даних з користувальницьких форм або інших інтерфейсів.

Атаки грубої сили (англ. Brute force) – дуже поширені проти мереж, оскільки вони, як правило, проникають в облікові записи зі слабкими комбінаціями імені користувача та пароля [10]. Остаточний сценарій був розроблений з метою отримання облікового запису SSH та MySQL шляхом запуску атаки грубої сили словника проти основного сервера.

Для виявлення потенційних активних атак, зазначених вище, використовують мережеві системи виявлення вторгнень – це програми, що розглядаються у якості вузлів мережі та пасивно перевіряють рух трафіку [4]. NIDS допомагає системним адміністраторам виявити порушення безпеки в організації мережі. Така система виявлення вторгнень відстежує рух в між різними вузлами виявляючи підозрілі шаблони у вхідних пакетах та шукаючи підозрілу активність, яка може бути атакою на систему або несанкціонованою діяльністю.

Деякі атаки можуть виникати та залишатися в локальній мережі або ініціюватися всередині мережі з цільовим об’єктом, що знаходиться поза мережею [11]. NIDS може контролювати вхідний, вихідний та місцевий трафік. Перевірка вихідного або місцевого руху може дати цінну інформацію про шкідливі дії, як і перевірка вхідного трафіку [4].

Великий сервер NIDS може бути налаштований у магістральну мережу, щоб контролювати весь трафік; або менші системи можуть бути налаштовані для контролю пакетів для певного сервера, комутатора, шлюзу або маршрутизатора [10].

Хоча розроблено багато NIDS, всі системи прагнуть функціонувати одним із двох способів: NIDS – це або системи, засновані на підписах, або на основі аномалії [5]. Обидва – це механізми, що відокремлюють звичайний мережевий трафік від зловмисних дій.

NIDS на основі підпису відноситься до виявлення атак шляхом пошуку конкретних шаблонів, таких, як послідовності байтів у мережевому трафіку або відомі зловмисні послідовності інструкцій, що використовуються зловмисними програмами [4]. Ця термінологія походить від антивірусного програмного забезпечення, де «підписи» – хеш від блоку коду специфічного для визначеної загрози, що досить часто використовується для позначення та ідентифікації відомих вірусів. Хоча такий тип NIDS може легко визначити відомі атаки, використовуючи такий підхід, важко виявити нові атаки, для яких не існує жодних баз даних або патернів.

Системи виявлення вторгнень на основі аномалії вперше були введені для виявлення невідомих атак, частково через швидкий розвиток шкідливих програм [4]. Основний підхід полягає у використанні методів машинного навчання для створення моделі, а потім порівняння нової поведінки з цією моделлю. Оскільки ці моделі можуть бути навчені відповідно до програм та конфігурацій апаратних засобів, метод на основі машинного навчання має кращу узагальнену властивість у порівнянні з традиційними IDS на основі підписів. Хоча такий підхід дозволяє виявити атаки для яких відсутні бази даних підписів, його використання може призвести до помилкових позитивних результатів: раніше невідома законна діяльність також може бути віднесена до зловмисної. Крім того, оптимальний алгоритм вибору властивостей робить процес класифікації, що використовується при виявленні, більш надійним. Головним недоліком NIDS на основі аномалій є час та ресурси, які будуть використані для навчання моделі, а також для функціонування самої системи, що може погіршити продуктивність. Але в цілому другий підхід є насьогодні значно більш точним та ефективним.

Основними операційними системами, на яких функціонують серверні компоненти сучасних NIDS, є системи, засновані на ядрі Linux [10]. Такі системи переважно є безкоштовними, мають відкритий вихідний код та безліч можливостей для налаштування. Тобто доцільно розглянути саме такі системи в якості цільових для розробки системи виявлення вторгнень.

Технології машинного навчання сьогодні розвиваються дуже активно та вже досить довго використовуються в різних сферах [13]. Вони показують більш високу ефективність, ніж класичні методи при вирішення багатьох завдань. Однією з таких є виявлення аномалій в даних. Тому вони все частіше використовуються в сфері інформаційної безпеки, наприклад для побудови систем виявлення вторгнень [12].

Глибинне навчання – це клас алгоритмів машинного навчання, який використовує кілька шарів штучних нейронів для поступового вилучення функцій вищого рівня із вихідних даних [7]. Саме поява та впровадження цих методів дала поштовх для розвитку машинного навчання останні кілька років. Існують багато

різновидів алгоритмів глибинного навчання – так званих архітектур. До них можна віднести глибинну мережу переконань, рекурентні нейронні мережі, згорткові нейронні мережі тощо. Такі методи досить ефективно застосовуються в таких галузях як комп'ютерний зір, обробка природної мови, розпізнавання звуку, виявлення аномалій в часових рядах, фільтрація соціальних мереж, а також активно впроваджуються у сфері інформаційної безпеки. Тобто можна зробити висновок, що є доцільним дослідити ефективність алгоритмів глибинного машинного навчання, у контексті завдання побудови системи виявлення вторгнень.

Інформаційні системи підтримують роботу в складноструктурованих предметних областях [14], та NIDS не є виключенням. Навіть сучасні алгоритми не можуть працювати без помилками, тому важливо надати можливість відображення інформації про трафік, яку було проаналізовано. Крім того, таким чином можна збільшити ефективність системи в подальшому, шляхом дослідження зібраних даних.

Зараз найбільш поширеним типом баз даних є реляційні бази [15]. Реляційна модель, яка використовується в таких базах даних, надає наступні переваги:

– простота і доступність для розуміння кінцевим користувачем (єдиною інформаційною конструкцією є таблиця);

– при проектуванні реляційних баз даних застосовуються суворі правила, що базуються на математичному апараті;

– реляційна модель забезпечує повну незалежність даних. При зміні структури реляційної бази даних зміни, які потрібно зробити в прикладних програмах, як правило, мінімальні, а також зазвичай достатньо прості у реалізації та тестуванні;

– маніпулювання даними на рівні мови системи управління базами даних (СУБД) проводиться ненавігаційно, тому для побудови запитів і написання прикладних програм немає необхідності знання конкретної організації бази даних у зовнішній пам'яті.

Крім вищезазначених переваг, реляційні бази даних мають широкий спектр функціоналу для агрегації та групування даних, відображення статистики, що робить їх доцільними для використання при побудові системи виявлення вторгнень.

1.2 Аналіз існуючих аналогів

На сьогоднішній день на ринку існує декілька основних систем виявлення вторгнень. Найвідомішими з них є Snort, Zeek та Suricata.

Snort, що належить Cisco Systems, є проектом з відкритим кодом і є безкоштовним для використання [16]. Це насьогодні одна з найбільш популярних систем для аналізу мережевого трафіку. Про це свідчить, що багато інших інструментів аналізу мережі були написані для інтеграції з результатами, які надає Snort. Крім того має досить активну спільноту. Приклад інтефейсу користувача Snort наведено на рисунку 1.1.

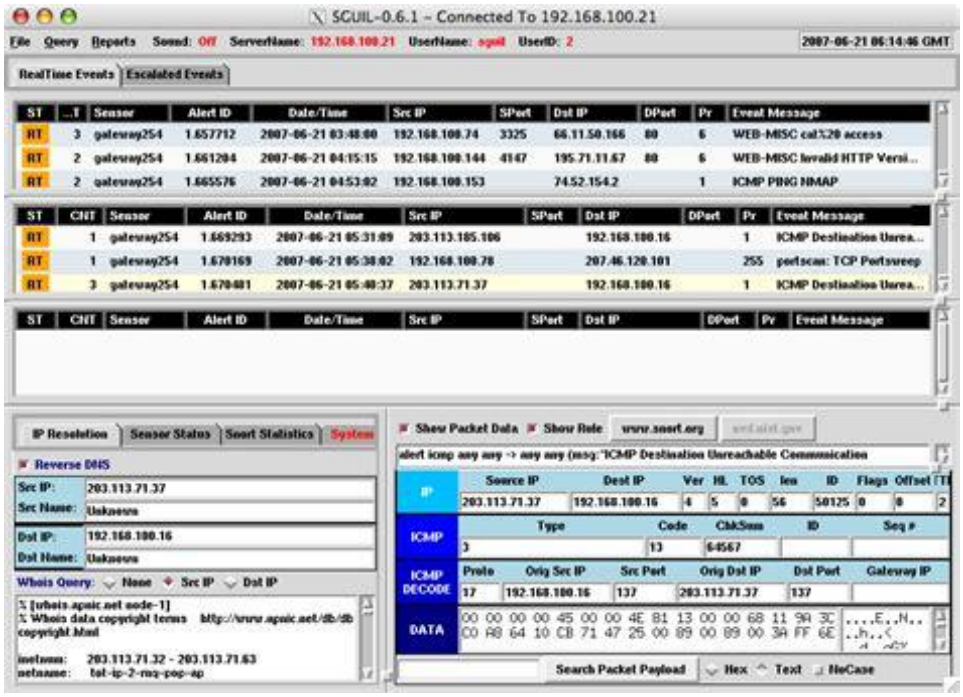


Рисунок 1.1 – Користувальницький інтерфейс Snort

Snort – фактично є системою фільтрації пакетів, яка збиратиме копії мережевого трафіку для аналізу. Однак інструмент має інші режими, і одним із них є виявлення вторгнень. У режимі виявлення вторгнень Snort застосовує «базові політики», тобто правила, які є основним структурним елементом умовних станів для виявлення зловмисних активностей. Snort може бути встановлений на майже всі популярні операційні системи: Windows, Linux та MacOS.

Zeek – це мережева система виявлення вторгнень, як і Snort. Цей безкоштовний NIDS працює одразу на декількох рівнях стандарту OSI, а також є досить популярним у науковців та спільнот у сфері інформаційної безпеки [17]. Загальний інтерфейс наведено на рисунку 1.2.

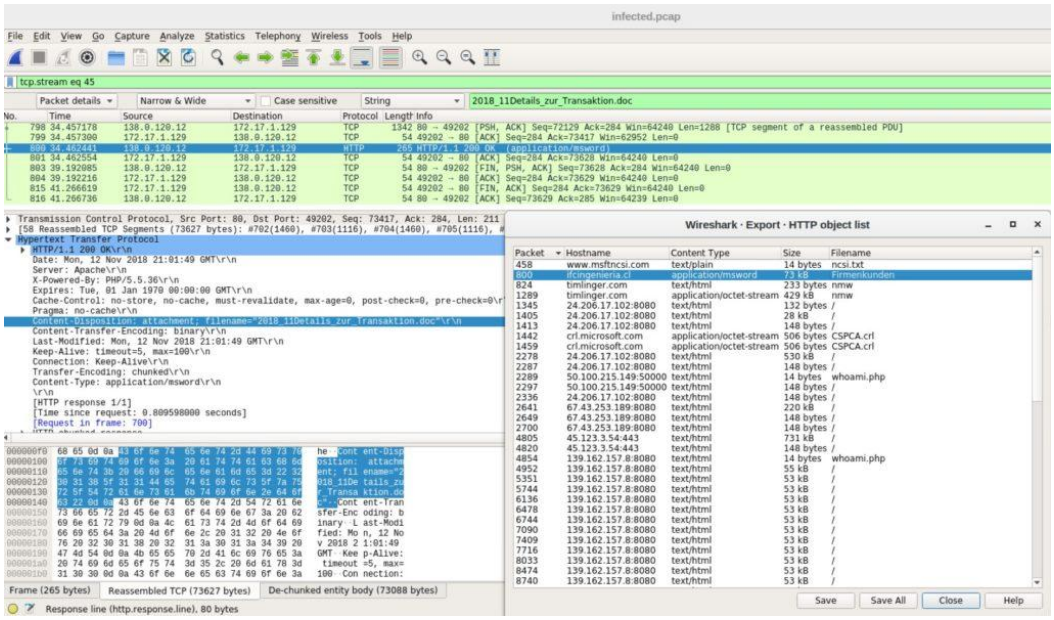


Рисунок 1.2 – Інтерфейс Zeek

Zeek використовує одночасно і систему підписів, і методи виявлення на основі аномалій. Цей додаток здатен розпізнавати шаблони рівня бітів, які вказують на шкідливу активність між пакетами.

Можна налаштувати виправні дії, які запускаються автоматично сценарієм тих чи інших правил. Програмне забезпечення можна встановити на Unix, Linux та MacOS. Zeek часто розглядають у парі з Dynamite-NSM – безкоштовним

монітором мережевої безпеки, який надає додаткового функціоналу для графічного відображенням журнальних даних системи.

Suricata – мережева система виявлення вторгнень, яка працює на прикладному рівні OSI. Це безкоштовний інструмент, який має подібні можливості, як і у Zeek [5]. Хоча Suricata працює на рівні програми, вона все ще маю можливість аналізувати дані більш глибоких рівнів, хоче з дещо більш обмеженим функціоналом. Наприклад є можливість отримати деталі пакета, що дозволяє програмі обробки отримувати інформацію на рівні протоколу з заголовків, яка включає наступну інформацію: тип шифрування, дані транспортного рівня та Інтернету.

Ця система також використовує методи виявлення на основі аномалій [18]. Окрім пакетних даних, інструмент може перевіряти сертифікати TLS, HTTP-запити та транзакції DNS, а також витягувати сегменти з файлів на рівні бітів для виявлення вірусів. Базовий інтерфейс Suricata наведено на рисунку 1.3.

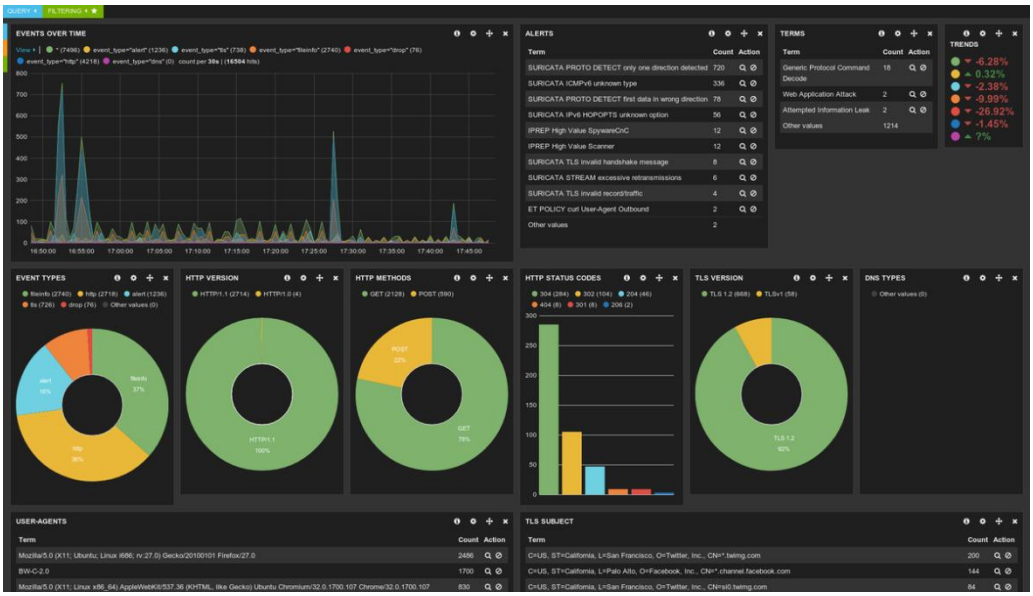


Рисунок 1.3 – Інтерфейс Suricata

Suricata – один із багатьох інструментів, сумісних зі структурою даних Snort. Він здатний реалізовувати базові правила Snort. Великою додатковою перевагою цієї сумісності є те, що спільнота Snort може відповісти на питання

щодо прийомів використання Suricata. Інші інструменти, сумісні з Snort, також можуть інтегруватися з Suricata.

Головним недоліком наведених вище мережевих систем виявлення вторгнень є складність розгортання на кластера з декількох серверів, бо фактично додаток, який є їх основною складовою частиною, має бути встановлений та налаштований окремо на кожен з ресурсів. Це також додає обмежень для аналізу інцидентів. Тобто доцільним є створення системи, яку можна було легко розгорнути на кластер з кількох серверів та об’єднати інциденти для аналізу.

1.3 Постановка задачі

Метою даної роботи є дослідження методів глибинного навчання для вирішення задачі побудови системи виявлення вторгнень.

У ході виконання дослідницької та практичної роботи магістру необхідно вирішити наступні завдання:

– провести аналіз предметної області мережевих систем виявлення вторгнень;

– провести аналіз методів глибинного навчання в контексті виявлення фактів кібератак на хмарні системи з метою вибору декількох з них для подальшого дослідження;

– підготувати або обрати та обробити датасет з даними моніторингу Linux-систем, у яких було змодельоване різноманітні кібератаки;

– спроектувати архітектуру прикладного додатку;

– спроектувати схему бази даних прикладного додатку;

– програмно реалізувати тренування моделей та систему їх порівняння для проаналізованих методів глибинного навчання;

– провести дослідження обраних методів глибинного машинного навчання в контексті рішення задачі розробки мережевої системи виявлення вторгнень;

– розробити варіант NIDS, в якому запроваджена найкращі з досліджених методів;

Отже, практичною задачею є розробка системи виявлення вторгнень на основі методів машинного навчання для серверів, які працюють на операційних системах, заснованих на ядрі Linux.

У якості технології для тренування моделей було використано мову Python та фреймворк для глибокого навчання tensorflow. Для додатку-агента, що збирає дані мережевого трафіку, було використано мову Python та бібліотеку scapy [19]. Для написання додатку, який розгортається на моніторинговий сервер, було використано Python фреймворк aiohttp на базі asyncio та реляційну СУБД PostgresSQL у якості базового сховища даних. Також було використано Vue.js для побудування клієнтської частини системи моніторингу.

Розроблений додаток повинен містити у собі наступний функціонал:

– перегляд подій у системі;

– відображення подій з окремих серверів;

– відображення списку підозрілих на аномалію подій;

– відображення інформації про підозрілих на аномалію подій;

– відображення додаткових даних про кожну з подій;

– сортування подій за різними критеріями;

– побудування стовпчастих діаграм подій згрупованих за різними критеріями;

– будування графіків аномалій в залежності від часу;

– нотифікація користувача у випадку надходження підозрілих інцидентів.

Система призначена для використання інженерами та системними адміністраторами та повинна бути досить зручною в використанні.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Призначення розробки

Система виявлення вторгнень – програмна система для забезпечення інформаційної безпеки інтернет ресурсів, яка аналізує мережевий трафік та виявляє підозрілі на загрози активності, а також сповіщає користувачів у разі появи аномальних патернів. Вона призначена в першу чергу для інженерів та адміністраторів, які мають намір розгорнути додатковий моніторинг безпеки мережі до своїх кластерів, розгорнутих на хмарних інфраструктурах різних провайдерів. Навіть найбільш захищені та добре спроектовані програмні продукти мають ризик бути атакованими зловмисником. Наслідки таких атак можуть бути дуже великими, як з точки зору втрати грошей, так і з точки зору репутації. Тому є важливим вчасно ідентифікувати потенційні загрози, щоб мати можливість якнайшвидше відреагувати та вжити заходи, яких потребує конкретна ситуація. Щоб забезпечити таку можливість, потрібно розробити систему, яка здатна виявляти підозрілі мережеві активності та доносити цю інформацію до відповідного користувача.

2.2 Вимоги до програмного продукту

Згідно з аналізом проблемної області, наведеним в розділі 1, система повинна складатися з наступних частин:

– агента, який збирає інформацію про трафік мережі на окремих ресурсах;

– серверу для моніторінгу, до якого надходять інциденти від агентів, а він, в свою чергу, класифікує та обробляє для подальшого зберігання;

– користувальницького інтерфейсу, основна функція якого є відображення даних з інцидентів в різних форматах.

Для розробки потрібно використовувати дві основних мови програмування: Python для реалізації частин для обробки даних та JavaScript написання користувальницького інтерфейсу. У якості середовища розробки слід використати багатофункціональний текстовий редактор від Microsoft – Visual Studio Code.

Агента, який збирає інформацію з окремих ресурсів, слід реалізовувати за допомоги Python бібліотеки scapy.

Для розробки ресурсу для моніторингу треба використати Python бібліотеку aiohttp у якості асинхронного API серверу, а також сервіси celery для реалізації протоколу асинхронних завдань та RabbitMQ для реалізації протоколу обміну повідомлень.

Користувальницький інтерфейс слід реалізовувати за допомогою JavaScript фреймворку Vue.js та бібліотеки для побудови діаграм d3.js.

Для тренування моделей глибинного навчання на основі даних мережевого трафіку треба використати Python бібліотеку tensorflow від компанії Google.

Система, що розробляється, повинна бути зручною в розгортанні, тому потрібно розробити модулі для налаштування компонентів за допомогою системи управління конфігураціями ansible.

Крім того, система повинна надавати функціонал нотифікації підозрілих інцидентів шляхом сповіщення у додатку для корпоративного обміну повідомлень – Slack.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Аналіз та UML-моделювання предметної області

Загрози кібербезпеки все частіше стають поширеними у світових компаніях, що призводить до значних втрат доходу та репутації бізнесу в цілому. Тому є важливим приділяти увагу захищеності програмних систем та забезпечувати рівень безпеки максимально можливим в даних конкретних умовах.

Атака вторгнення в компоненти програмної системи може завдати серйозної шкоди вашим мережам та інтегрованим системам. Тому є доцільним впровадження мережевої системи виявлення вторгнень, що є ефективним рішенням безпеки. Вона попередньо аналізує, виявляє та попереджає користувачів про підозрілі дії у мережі.

NIDS читає всі вхідні пакети та шукає будь-які підозрілі зразки. Коли виявляються загрози, виходячи з її серйозності, система може вжити таких дій, як сповіщення адміністраторів або заборона доступу до мережі вихідної IP-адреси [5].

Мережева система виявлення вторгнень здебільшого розміщується в стратегічних точках мережі, щоб мати можливість стежити за трафіком, який прямує до або з різних пристроїв цієї мережі [9]. Окрім моніторингу вхідного та вихідного мережевого трафіку, сервер NIDS також може сканувати системні файли, які шукають несанкціоновану діяльність, підтримувати цілісність даних і файлів та виявляти зміни в основних компонентах і налаштуваннях сервера.

З точки зору користувача основним завданням системи виявлення вторгнень є відображення даних, які було оброблено з метою подальшого аналізу, а також сповіщення про всі підозрілі інциденти, які проходили в мережі.

Також слід зазначити, що хмарні кластери можуть складатися з ресурсів, за які відповідають різні команди. Ці команди навіть можуть працювати на різні організації. Тому має сенс надати можливість розділення привілеїв NIDS: звичайний користувач може переглядати тільки інформацію, про сервери, які

належать йому [20]. Взявши до уваги всю інформацію, згадану вище, було побудовано діаграму прецедентів, яку зображено на рисунку 3.1, яка дозволяє візуалізувати основний функціонал системи виявлення вторгнень, що розробляється.

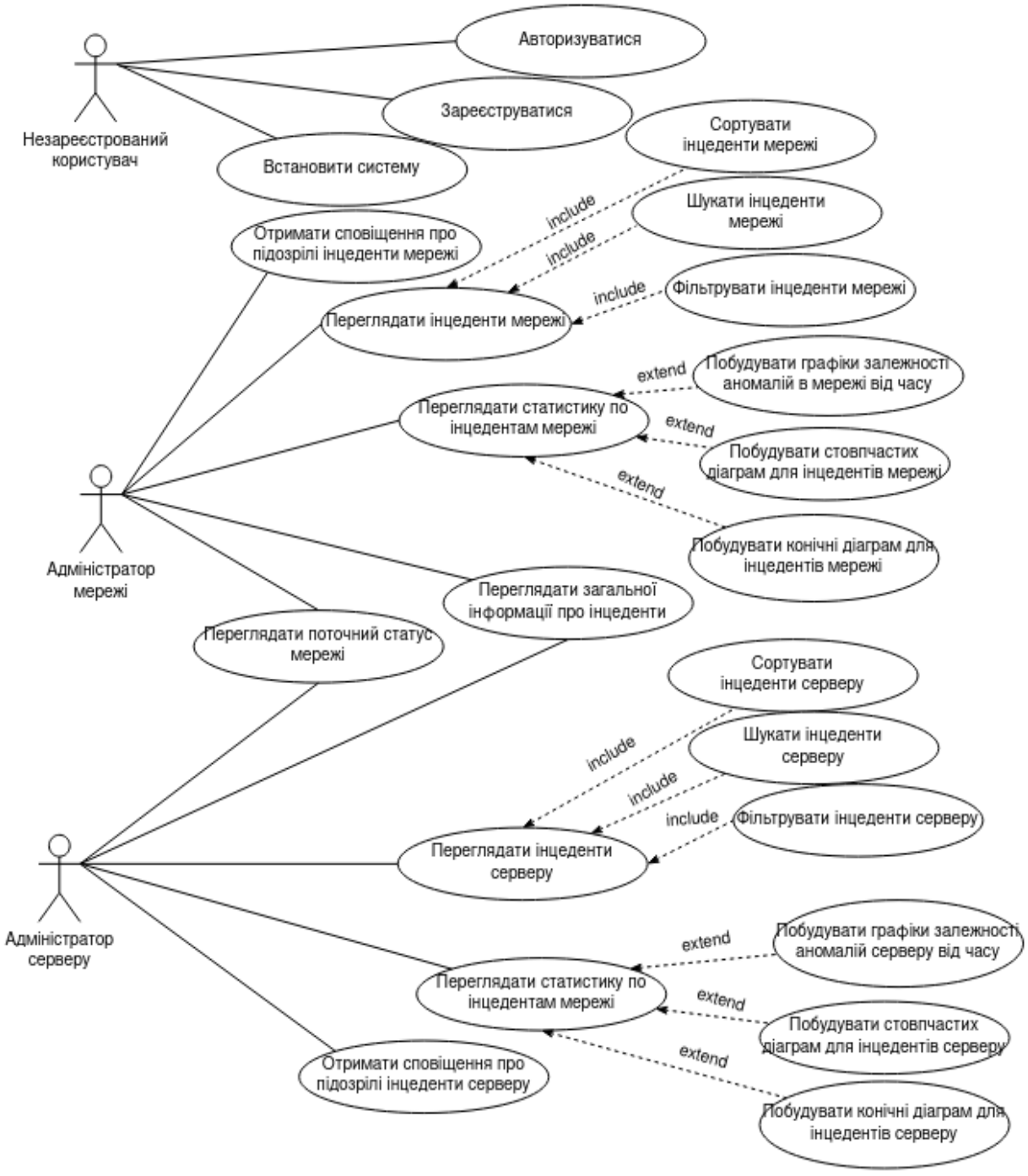


Рисунок 3.1 – UseCase-діаграма

З наведеної діаграми видно, що система має три категорії користувачів: неавторизовані користувачі, адміністратори мережі та адміністратори серверу.

Неавторизований користувач може встановити систему, а також зареєструватися та авторизуватися. Адміністратори мережі можуть отримувати сповіщення про підозрілі події в системі, переглядати статистики по інцидентам та аномаліям, а також виконувати моніторинг поточного статусу мережі та переглядати додаткові дані з інцидентів. Адміністратор серверу може отримувати сповіщення про аномалії, які пов’язані безпосередньо с сервером, який у нього під контролем, переглядати інформацію про інциденти сервера та його підозрілі логи, а також, як і адміністратор мережі, переглядати її поточний статус та додаткові дані по інцидентах.

3.2 Огляд методів машинного навчання

Машинне навчання – це різновид комп’ютерних алгоритмів, які автоматично вдосконалюються завдяки досвіду [21]. Алгоритми машинного навчання будують математичну модель на основі даних – зразків, для того, щоб робити прогнози або приймати рішення для інших наборів даних цього типу.

Основними методами машинного навчання є:

- регресійний аналіз;
- байєсівська мережа;
- машина опорних векторів;
- штучні нейронні мережі.

Регресійний аналіз – це сукупність статистичних методів для оцінки взаємозв’язків між залежною змінною та однією або кількома незалежними змінними. Найпоширенішою формою регресійного аналізу є лінійна регресія, при якій дослідник знаходить ту лінію (або більш складну лінійну комбінацію), яка найбільш відповідає даним згідно з конкретним математичним критеріям.

Байєсівська мережа (також відома як мережа Байєса, мережа переконань або мережа прийняття рішень) – це імовірнісна графова модель, яка представляє

набір змінних та їх умовних залежностей за допомогою спрямованого ациклічного графу. Байєсові мережі добре підходять для того, щоб для даної події, що сталася, передбачати ймовірність того, що якась із кількох можливих відомих причин була основним фактором, що сприяє цій події. Наприклад, байєсівська мережа може представляти ймовірнісні зв’язки між захворюваннями та симптомами. Враховуючи симптоми, мережа може використовуватися для обчислення ймовірності наявності різних захворювань.

Сутність методу машини опорних векторів (англ. SVM) полягає в тому, що ділить ряд об’єктів на класи таким чином, що максимально широка область навколо меж класу залишається вільною від об’єктів [22]. Основна ідея – переклад вихідних векторів в простір більш високої розмірності й пошук гіперплощині, що розділяє з максимальним зазором в цьому просторі.

Штучні нейронні мережі – це обчислювальні системи, які базуються на структурі мозку тварин [21]. Кожен нейрон є вузлом, який з'єднаний з іншими за допомогою зв'язків, які мають вагу, що фактично визначає силу впливу одного вузла на інший. Саме в визначенні значень важелів кожної ланки з’єднання і полягає навчання нейронної мережі.

Останні кілька років активно розвиваються методи глибинного навчання – клас алгоритмів машинного навчання, який використовує кілька шарів нейронів для поступового вилучення функцій вищого рівня із вихідних даних [21]. Наприклад, при обробці зображень перші шари можуть ідентифікувати краї, тоді як останні шари можуть ідентифікувати такі поняття, що стосуються людини, такі як цифри, літери чи грані.

Одним з самих розповсюджених методів глибинного навчання є згорткові нейронні мережі [23]. Назва «згорткова нейронна мережа» означає, що в мережі використовується математична операція, яка називається згорткою – спеціалізований вид лінійних операцій. Такі мережі використовують згортку замість загального множення матриць принаймні в одному зі своїх шарів.

Ще одним варіантів архітектури глибинного навчання є рекурентні нейронні мережі, що являють собою вид нейронних мереж, де зв'язки між

елементами утворюють спрямовану послідовність [21]. Завдяки цьому з'являється можливість обробляти серії подій у часі або послідовні просторові ланцюжки. Рекурентні мережі можуть використовувати свою внутрішню пам'ять для обробки послідовностей довільної довжини. Найбільш ефективними є наступні різновиди рекурентні нейронних мереж: Довга короткострокова пам'ять та керовані рекурентні блоки.

Довга короткострокова пам'ять (LSTM) – це штучна рекурентна нейромережева архітектура, що використовується в галузі глибинного навчання [21]. На відміну від стандартних рекурентних нейронних мереж, LSTM має зв'язки в зворотному напрямку. Він може обробляти не лише окремі точки даних (наприклад, зображення), а й цілі послідовності даних (наприклад, мова або відео). Наприклад, LSTM застосовний до таких завдань, як розпізнавання рукописного тексту.

Керовані рекурентні блоки – це вентильний механізм у рекурентних нейронних мережах, представлений у 2014 році [21]. Він схожий з методом LSTM, але мають менше параметрів. Ефективність GRU щодо певних завдань моделювання поліфонічної музики, моделювання мовного сигналу та обробки природних мов виявилася подібною до результатів роботи LSTM. Показано, що GRU демонструють ще більшу ефективність на деяких менших та менш частих наборах даних.

Слід зазначити, що саме методи глибинного навчання є найбільш ефективними для широкого класу завдань пов’язаних з класифікацією інформації. Згорткові нейроні мережі здебільшого використовують у сфері комп'ютерного зору, наприклад для виявлення об’єктів на зображеннях, або у якості допоміжних методів для виявлення параметрів з неструктурованої інформації. Але дані трафіку здебільшого є текстовими або числовими, тому є доцільним розглядати саме методи рекурентних нейронних мереж для реалізації системи виявлення вторгнень.

Таким чином, були проаналізовані основні методи машинного навчання та обрано два основних методи для подальшого аналізу.

3.3 Аналіз обраних архітектур

Важливим елементом будь-яких методів машинного навчання є функція активації. Вона може бути кінцевим або проміжним елементом нейронних мереж, який перетворює дані з набору різноманітних параметрів у кінцевий результат, який або буде фінальним результатом побудованої моделі, або буде готовий для передачі на наступний шар мережі. Одними з найбільш розповсюджених є наступні функції активації: логістична, гіперболічний тангенс, нормована експоненційна функція.

Логістична функція $\sigma(x)$ повертає результат в інтервалі $(0, 1)$ та обчислюється за наступною формулою:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

Гіперболічний тангенс $tanh(x)$ коливається в межах $(-1, 1)$ та визначається використовуючи формулу 3.2:

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

Як відомо, рекурентні мережі можна розглядати, як класичні нейроні мережі розгортаються на декілька ітерацій, кожна з яких приймає у якості параметрів безпосередньо дані для навчання, а також результат попередньої ітерації. LSTM архітектура дозволяє підтримувати стан, який накопичено з попередніх ітерацій, так звану «пам’ять». Він дозволяє приймати рішення на основі попередніх результатів. Сам стан поділяється на короткочасний та довгочасний. Клітина LSTM складається з наступних компонентів [21]:

– $g_{(t)}$ – є результатом обробки вхідних даних на поточній ітерації;

– $c_{(t)}$ – підтримує довготривалий стан клітини;

– $f_{(t)}$ – обчислює які елементи з попереднього стану, які не будуть передаватися на наступну ітерацію, тобто «забуті»;

– $i_{(t)}$ – визначає, яка частина $g_{(t)}$ додається до довготривалого стану;

– $o_{(t)}$ – обчислює, яка частина довготривалого стану буде додана до результату;

– $h_{(t)}$ – короткочасна пам'ять, яка фактично є кінцевим результатом.

Позначимо $W_{xi}, W_{xf}, W_{xo}, W_{xg}$ – матриці важелів відповідних компонентів та їх комбінацій з кожним параметром з вхідних даних, $W_{hi}, W_{hf}, W_{ho}, W_{hg}$ – аналогічні матриці для комбінацій компонентом, який відповідає за короткочасну пам'ять, а b_i, b_f, b_o, b_g – вектори зсуву для кожного з компонентів, що визначають вихідне результуюче значення, якщо кожен з вхідних параметрі дорівнюють нулю. Загалом, можна описати процес роботи клітини LSTM використовуючи формулу 3.3:

$$
\begin{aligned}
i_{(t)} &= \sigma\left({W_{xi}}^T x_{(t)} + {W_{hi}}^T h_{(t-1)} + b_i\right), \\
f_{(t)} &= \sigma\left({W_{xf}}^T x_{(t)} + {W_{hf}}^T h_{(t-1)} + b_f\right), \\
o_{(t)} &= \sigma\left({W_{xo}}^T x_{(t)} + {W_{ho}}^T h_{(t-1)} + b_o\right), \\
g_{(t)} &= \tanh\left({W_{xg}}^T x_{(t)} + {W_{hg}}^T h_{(t-1)} + b_g\right), \\
c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)}, \\
h_{(t)} &= o_{(t)} \otimes \tanh\left(c_{(t)}\right)
\end{aligned}
\qquad (3.3)
$$

Для GRU, на відміну від LSTM, $c_{(t)}$ та $h_{(t)}$ об'єднані до одного компоненту, якій відповідає з стан мережі [21]. Крім того, за $f_{(t)}$ та $i_{(t)}$ разом відповідає $z_{(t)}$. Якщо $z_{(t)} = 1$, то цей елемент буде пропущено, а якщо 0, то додано до поточного стану. Також GRU замість $o_{(t)}$ має компонент $r_{(t)}$, який контролює яку саме частину попереднього стану передавати до $g_{(t)}$. Таку архітектуру клітини можна зобразити за допомогою формули 3.4.

$$z_{(t)} = \sigma\left({W_{xz}}^T x_{(t)} + {W_{hz}}^T h_{(t-1)} + b_z\right),$$
$$r_{(t)} = \sigma\left({W_{xr}}^T x_{(t)} + {W_{hr}}^T h_{(t-1)} + b_r\right),$$
$$g_{(t)} = \tanh\left({W_{xg}}^T x_{(t)} + {W_{hg}}^T (r_{(t)} \otimes h_{(t-1)}) + b_g\right),$$
$$h_{(t)} = z_{(t)} \otimes h_{(t-1)} + \left(1 - z_{(t)}\right) \otimes g_{(t)} \quad (3.4)$$

Тобто GRU є спрощеною версією ніж LSTM, але при цьому, перезаписує дані стану для визначеного параметру.

3.4 Адаптація методу класифікації трафіку

Використання комбінації різних архітектур нейронних мереж часто дозволяє покращити ефективність моделей. В даній роботі запропоновано використати LSTM та GRU мережі разом з методом машини опорних векторів.

Основна мета машини опорних векторів – розділити групу елементів, які розглядаються, як вектори *p*-вимірного простору на дві частини за допомогою *(p-1)*-вимірної гіперплощини. Таких гіперплощин може бути нескінченно багато, але ідея полягає в тому, щоб обрати саме ту площину, яка буде найбільш віддалена від найближчих елементів з обох боків. Приклад класифікації для двомірного простору наведено на рисунку 3.2.

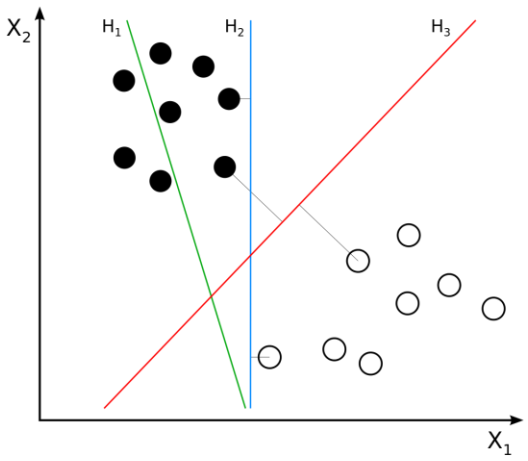


Рисунок 3.2 – SVM для двомірного простору

Такий метод можна застосувати, як останній етап рекурентної нейронної мережі. З кожного нейрона на попередньому шарі ми отримаємо значення вихідних параметрів, які об'єднаємо у вихідний результат моделі. Тобто для LSTM-SVM мережі отриману архітектуру можна описати згідно з рисунком 3.3:

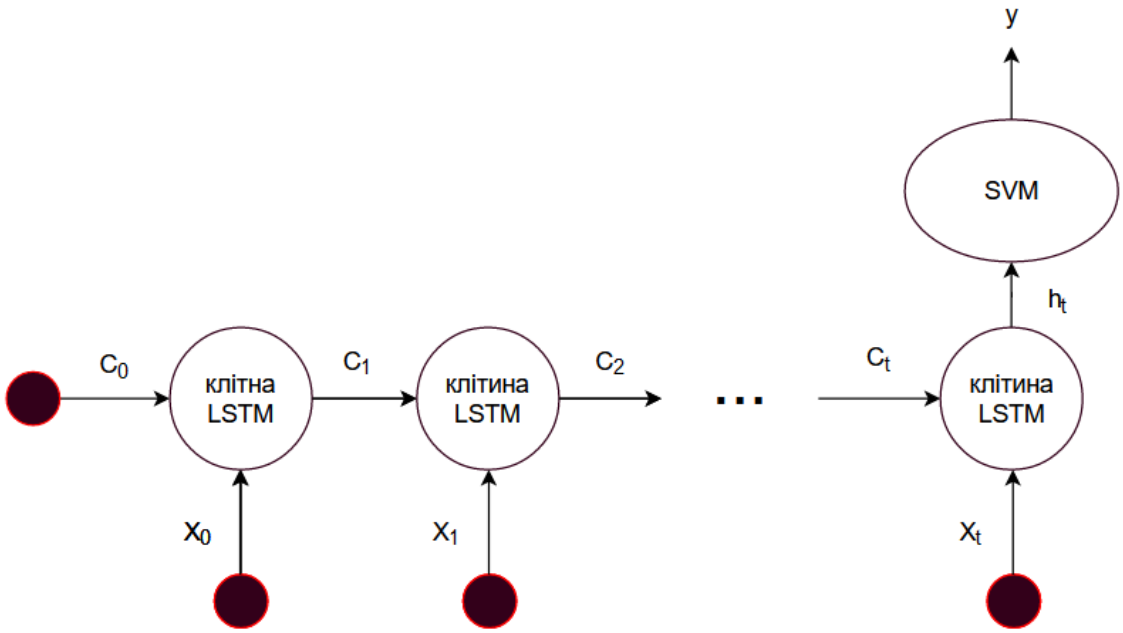


Рисунок 3.3 – Архітектура LSTM-SVM

Позначимо w, як матрицю важелів, а x_i, y_i – вхідні та вихідне значення, що очікується, для i тренувального прикладу, а n – кількість прикладів для тренування. Функція втрат для методу машини опорних векторів зазвичай описується за допомогою формули 3.5:

$$J(w,b) = \frac{1}{2}w^T w + C\sum_{i=1}^{n} \max\left(0, 1 - y_i'(w^T x_i + b_i)\right)^2 \qquad (3.5)$$

Але в архітектурі, що розглядається на вхід машині опорних векторі буде передаватися результат роботи рекурентної мережі, де ми маємо набір важелів $\boldsymbol{W}$ та результат навчання h_i. Тобто цільова функція матиме вигляд згідно формули 3.6:

$$J(\boldsymbol{W}, b) = \frac{1}{2}\boldsymbol{W} + C\sum_{i=1}^{n} \max(0, 1 - y_i'(h_i\boldsymbol{W} + b_i))^2 \quad (3.6)$$

Враховуючи доданий шар, натренована модель буде повертати пару з двох чисел: додатного, що визначає клас, до якого належать вхідні значення, та від'ємного. Функція $argmax$ повертає нам індекс максимального елементу з вектора, який передається, тобто в цьому випадку буде дорівнювати номеру класу, який модель визначила як правильний. Позначивши множину вхідних даних за x, функцію передбачення через $F(x)$, а визначений тип (загроза чи ні) за $result$, можна описати результат використовуючи формулу 3.7.

$$result = argmax(F(x)) \quad (3.7)$$

Слід зазначити, що для GRU-SVM побудована архітектура та обчислення будуть аналогічними. Після активації нейронів GRU на останньому етапі дані буде оброблено використовуючи класифікатор SVM, який сформує результат на даному етапі навчання.

3.5 Проектування бази даних

Система виявлення вторгнень обробляє великий об'єм інформації про мережевий трафік. Щоб мати змогу зберігати ці дані, з метою їх додаткового аналізу та дослідженню у майбутньому, буде використовуватися реляційна база даних, яка є зазвичай невід'ємною частиною кожного сучасного додатку, який має досить високі навантаження.

Процес проектування схеми даних є дуже важливим етапом при створенні будь якої програмної системи. База даних може зберігати велику кількість

систематизованої інформації, і надавати її у вигляді відповіді на запит клієнту, яким може бути як кінцевий користувач, так й інший сервіс, що в взаємодіє з додатком.

Проектування бази даних являє собою складний трудомісткий процес відображення предметної області у внутрішню модель даних [24]. Сам процес проектування бази даних полягає в створенні схеми бази даних і визначенні необхідних обмежень цілісності інформації, які потрібні для нормального функціонування додатку [15].

В якості моделі зберігання даних була обрана реляційна модель. Реляційна модель являє собою сукупність даних, що складається з набору двовимірних таблиць.

Така модель є зручною і найбільш звичною формою представлення даних. Будь-яка таблиця в реляційній базі складається з рядків, які називають записами, і стовпців, які називають полями. На перетині рядків і стовпців знаходяться конкретні значення даних [24].

Реляційна модель зберігання даних має наступні переваги:

– виклад інформації в простій і зрозумілій для користувача формі (таблиця);

– реляційна модель даних заснована на суворому математичному апараті, що дозволяє лаконічно описувати необхідні операції над даними;

– незалежність даних від зміни в програмному забезпеченні при зміні;

– для роботи з моделлю даних немає необхідності повністю знати організацію бази даних.

Однак, вона має і свої недоліки:

– відносно повільний доступ до даних;

– труднощі при створенні бази даних, заснованої на реляційній моделі;

– труднощі в перенесенні в таблицю складних відносин;

– потрібен відносно великий обсяг пам'яті.

Спочатку необхідно визначити основні сутності для зберігання даних системи виявлення вторгнень. До основних сутностей можна віднести: користувач, сервер, інцидент.

Сервер може містити наступні атрибути: ідентифікатор, IP-адресу, тип операційної системи, ім’я домену, використаний хмарний провайдер, час останньої активної дії.

До основних атрибутів інциденту можна віднести: унікальний ідентифікатор, тип інциденту, результат сканування, вхідну IP-адресу, час надходження інциденту, ім’я, вірогідність того, що інцидент є загрозливим, ступінь загрози.

Користувача характеризують наступні атрибути: ідентифікатор, електронна пошта, ім’я та пароль.

Сервер та інцидент зв’язані відношенням один до багатьох. При цьому у якості зовнішнього ключа таблиць використовується унікальний ідентифікатор серверу.

Користувач та сервер зв’язані між собою відношенням багато до багатьох. У якості додаткової сутності, яка реалізує даний тип відношення використовуються Користувач-Сервер.

Також для визначення ролі користувача, яка в подальшому використовується для надавання йому відповідних прав та привілеїв введена сутність – Роль, яка має атрибути ім’я та тип, та пов'язана з сутністю користувача відношенням один до багатьох.

Загальну схему бази даних, що була спроектована можна побачити на рисунку 3.4.

Таким чином, було спроектовано та побудували реляційну модель бази даних для мережевої системи виявлення вторгнень.

3.6 Розробка архітектури системи

В ході планування та проектування мережевої системи виявлення вторгнень було визначено, що система повинна складатися з наступних складових:

– агенту, який безпосередньо встановлюють на кожний сервер окремо; він збирає інформацію про мережевий трафік та надсилає її на моніторинговий сервер;

– моніторингового серверу, який розгортається на окремій машині та займається обробкою, аналізом та зберіганням даних з агентів та сповіщенням користувачів у випадку аномальних активностей в мережі;

– сховища даних, яке буде використовувати PostgreSQL у якості реляційної СУБД;

– користувацький клієнт, в якому за допомогою браузера користувач може переглядати інформацію про проаналізований трафік в різних форматах подання інформації.

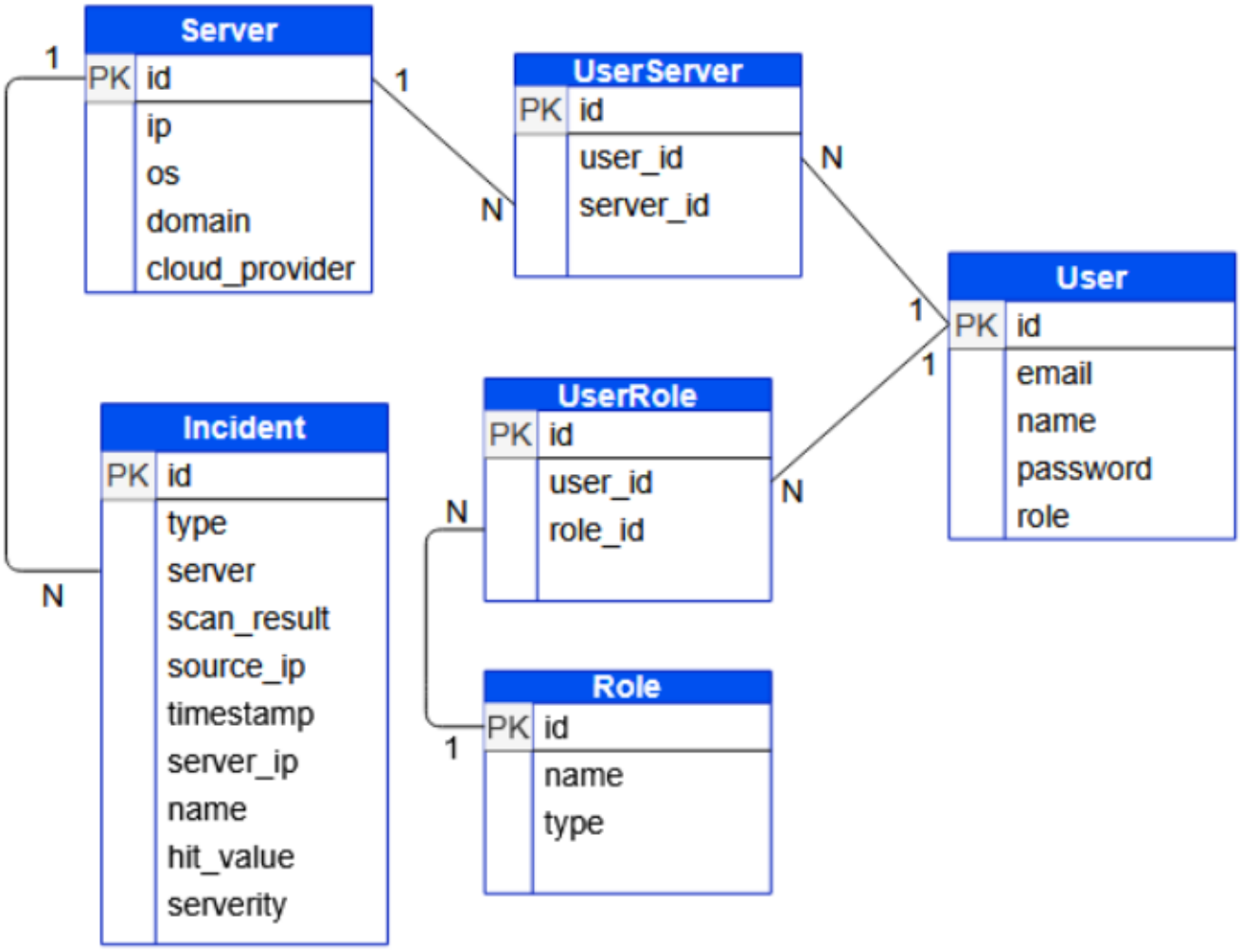


Рисунок 3.4 – Схема бази даних

Для реалізації моніторингового серверу було обрано мікросервісну архітектуру. архітектурний стиль, за яким єдиний додаток будується як сукупність невеличких сервісів, кожен з яких працює у своєму власному процесі та спілкується з рештою, використовуючи прості та швидкі протоколи передачі даних, зазвичай HTTP, але допускається такоє інші методи .

До основних переваг такого підходу можна віднести наступні фактори: компоненти є незалежними один від одного, кожен сервіс може бути протестований окремо без зайвих накладних витрат, а також з’являється можливість розгортати кожен з елементів додатку у різні періоди часу, використовуючи різні інструменти. З огляду на вимоги, що сформовані у попередніх розділах, моніторинговий сервер буде складатися з наступних компонентів-сервісів:

– API для агентів, основна мета якого фільтрування непотрібних запитів, їх попередня обробка та надсилання на наступний компонент для обробки та визначення типу інциденту;

– сервісу, який приймає дані з API для агентів та відповідає за аналіз зберігання даних;

– брокера повідомлень, який передає інформацію з першого компоненту на другий та навпаки;

– API, який використовує користувальницький клієнт для доступу до даних зі сховища;

– користувацький клієнт, в якому за допомогою браузера користувач може переглядати інформацію про інциденти, що були надіслані та проаналізований трафік у різних форматах.

Для опису загальної архітектури системи побудуємо діаграму розгортання – схему, яка призначена для візуалізації елементів і компонентів додатків, що існують лише на етапі її виконання, до яких відносяться виконані файли, динамічні бібліотеки, таблиці бази даних і т.д. Ті компоненти, які не використовуються на етапі виконання (наприклад, вихідні тексти програм), на діаграмі не відображаються [25]. Основні цілі, які переслідуються у процесі

будування діаграми розгортання для мережевої системи виявлення вторгнень: зображення різних частин системи, розподіл компонентів по її фізичним вузлам, та ілюстрація типів комунікації між різними ресурсами системи на етапі виконання.

Діаграма розгортання системи виявлення вторгнень, що розробляється, наведена на рисунку 3.5.

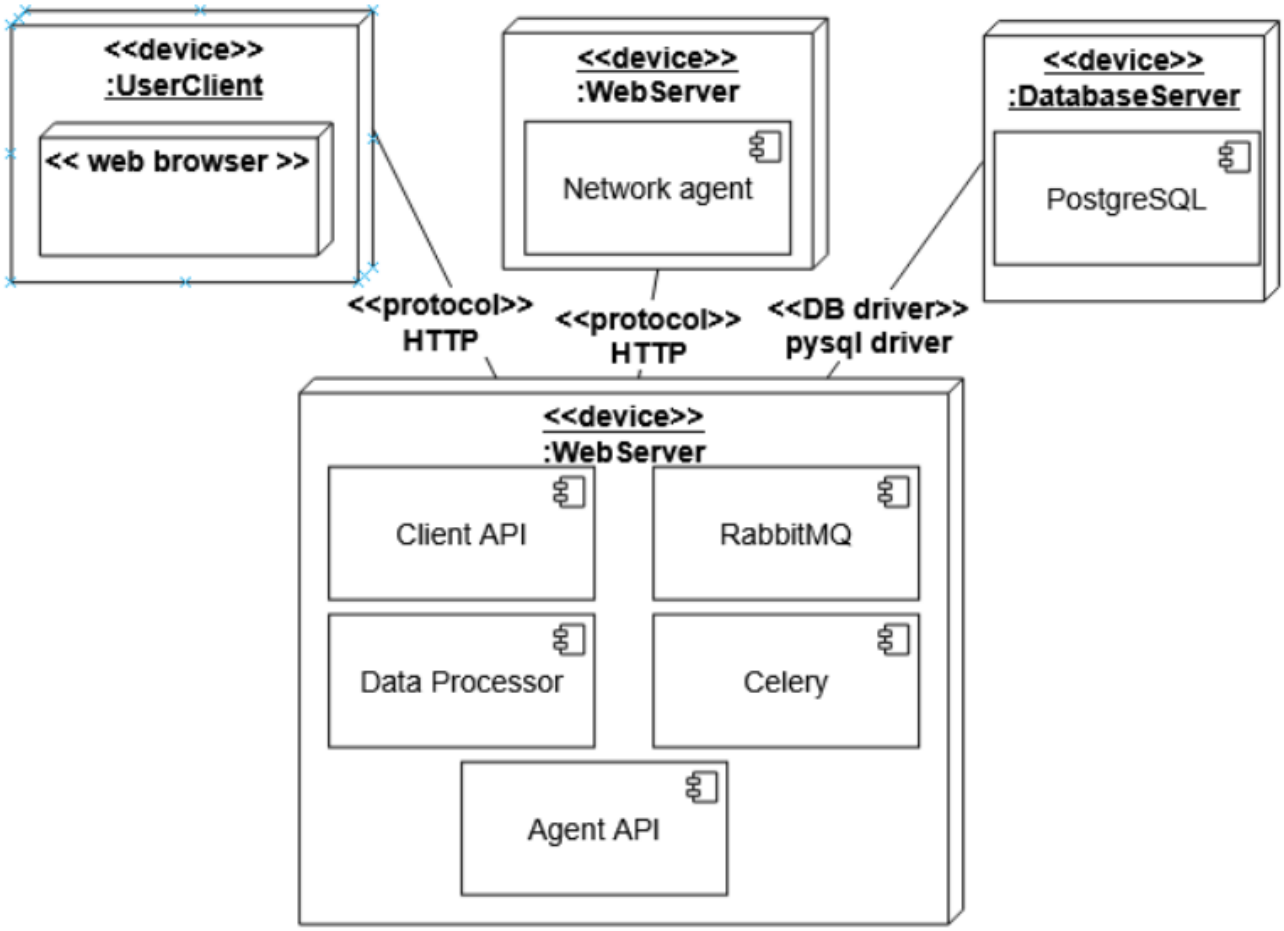


Рисунок 3.5 – Діаграма розгортання

Діаграма розгортання не демонструє зв’язки між компонентами моніторингового серверу, тому для демонстрації цих зв’язків побудуємо діаграму компонентів [20]. Схема компонентів розбиває фактичну систему, що розробляється, на різні високі рівні функціональності. Кожен відповідає за одну чітку задачу в рамках всієї системи і взаємодіє лише з іншими необхідними елементами за допомогою протоколів комунікації [24].

Діаграма компонентів дозволяє підтвердити, що необхідна функціональність системи відповідає вимогам. Ці схеми також використовуються як інструмент комунікації між розробником та зацікавленими сторонами системи. Програмісти та розробники використовують діаграми для формалізації плану для реалізації, що дозволяє краще приймати рішення про призначення завдань або необхідні вдосконалення навичок. Системні адміністратори можуть використовувати схеми компонентів для планування заздалегідь, використовуючи вигляд логічних програмних компонентів та їх взаємозв'язки в системі.

Опишемо складові частини системи. API агенту, який приймає запити з окремих серверів, після попередньої обробки надсилає ці дані до брокера повідомлень RabbitMQ, використовуючи модуль, який виконує роль сервісу повідомлень. Крім того, цей компонент містить контролер запитів та систему аутентифікації для агентів, для забезпечення конфіденційності та цілісності інформації.

Сервіс, який відповідає за обробку інцидентів, фактично є основним компонентом бізнес-логіки системи. Він має такі елементи: інтерфейс нетренованої моделі, яка виявляє аномальний трафік, класи, які є ORM обгорткою над сутностями бази даних, сервіс для сповіщення користувачів про підозрілі події в системи, контролер повідомлень, які надходять з агентів та модулі для обробки даних перед аналізом та зберіганням.

Цей сервіс взаємодіє з брокером повідомлень RabbitMQ, сервісом для асинхронних завдань Celery, який буде використано для відправлення повідомлень користувачу, а також з СУБД PostgreSQL для операції з реляційним сховищем.

Ще одним компонентом системи є сервіс клієнтського API, який надає інтерфейс до даних для браузерного клієнту системи. Він складається з контролеру запитів браузерних клієнтів, сервісу аутентифікації та модулей для зчитування даних зі сховища. Даний компонент взаємодіє з СУБД PostgreSQL.

Побудовану діаграму компонентів мережевої системи виявлення вторгнень наведено на рисунку 3.6.

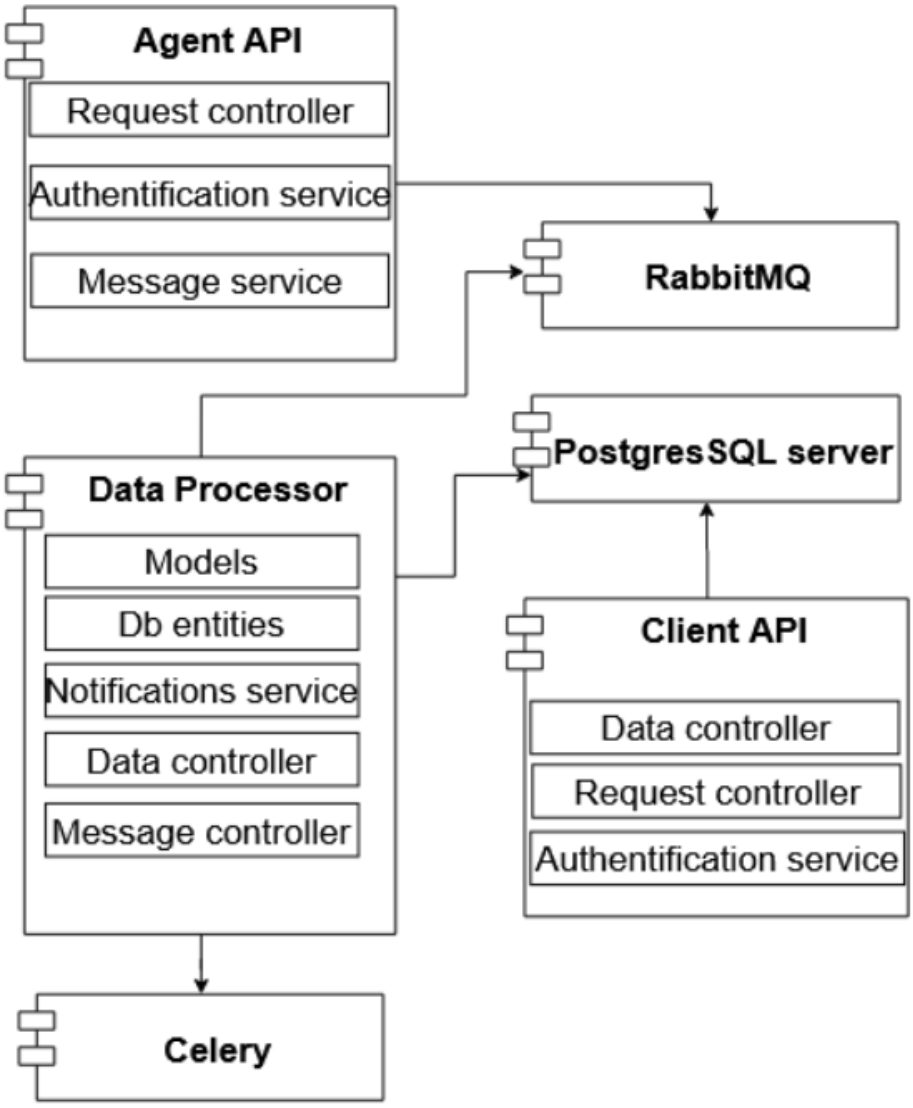


Рисунок 3.6 – Діаграма компонентів

Таким чином, було проведено аналіз предметної області, визначені нейронні мережі для дослідження, які були проаналізовані, натреновані та протестовані на підготовлених даних. Також, спроектовано архітектуру програмної системи, побудували діаграму розгортання, продемонстрували зв’язки між компонентами моніторингового серверу побудувавши діаграму компонентів.

4 ОПИС ДОСЛІДЖЕННЯ ОБРАНИХ МЕТОДІВ

4.1 Підготовка даних для дослідження

У якості базового датасету обрано набір даних, опублікований Канадським університетом з кібербезпеки [26]. Його було розроблено шляхом підняття кластеру серверів та емуляції найбільш поширених атак та збирання даних мережевого трафіку впродовж п’яти днів.

Інформацію в датасеті була надано у форматі pcap [19] – стандартному для програм моніторингового трафіку систем Linux. Такий формат не підходить для тренування моделі, бо його не підтримують бібліотеки машинного навчання. Тому кожен з файлів перетворено в формат csv використовуючи Python бібліотеку відкритим вихідним кодом – pyshark.

Наступним етапом було оброблено датасет для навчання тобто видалені ті параметри, які по тим чи іншим ознакам на є доцільними для тренування. Дані підготували за допомогою алгоритму, який наведено на рисунку 4.1. Спочатку всі параметри, які є нерелевантними, однаковими для всіх випадків або є числами з рухомою комою та майже не відрізнялися було видалено з датасету.

Після цього над отриманими даними застосували метод головних компонент, головне завдання якого – зменшити розмірність даних, тобто видалити параметри, розподіл значень яких не має сенсу використовувати для тренування.

Метод головних компонент дає можливість за m – числом початкових ознак виділити r головних компонент, або узагальнених ознак. Математична модель методу головних компонент базується на логічному припущенні, що значення множини взаємозалежних ознак породжують деякий загальний результат. Для реалізації даного методу було застосовано Python бібліотеку для машинного навчання scikit-learn, яка була розроблена на основі високоефективних інструментів для проведення математичних операцій над матрицями scyPy та numPy.

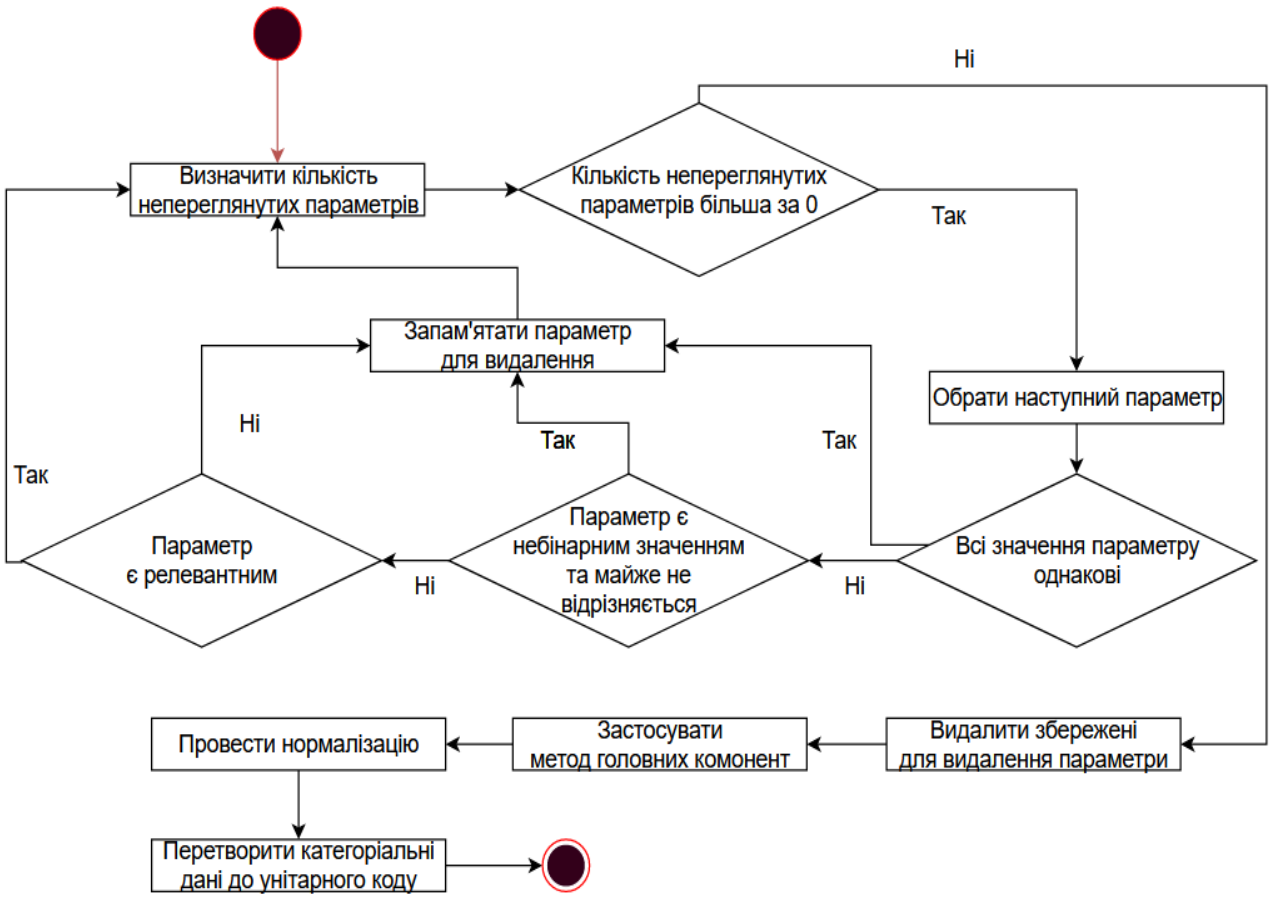


Рисунок 4.1 – Алгоритм підготовки дата сету

Далі було проведено нормалізацію даних за допомогою масштабування мінімуму та максимуму [14]. Тобто всі значення параметрів, які є числами з рухомою комою були перетворені в відповідні числа від нуля до одиниці, використовуючи формулу 4.1 (x' – отримане значення):

$$x' = \frac{x - min\ (x)}{max(x) - min\ (x)} \tag{4.1}$$

На останньому етапі було перетворено категоріальні значення до унітарного коду [7]. Нехай в нас є параметр, який має тільки два значення, наприклад назва протокол: TCP або UDP. Бібліотеки машинного навчання здебільшого можуть працювати тільки з числовими характеристиками або з двійковими значеннями. Тому доцільно розбити параметри такого типу на два бінарних: якщо йдеться

мова про протокол TCP – у відповідний параметр буде дорівнювати одиниці, а параметр для UDP – нуль, та навпаки.

Після обробки датасет складається з 43 параметрів. Деякі з параметрів та їх значень параметрів наведені в таблиці 4.1.

Таблиця 4.1 – Результати дослідження

same_srv_rate	diff_srv_rate	srv_diff_host_rate	dst_host_count	dst_host_srv_count
1,00	0,00	0,20	1	250
0,02	0,05	0,00	0,76	0,03
1,00	0,00	0,00	0,76	0,04
1,00	0,00	0,02	1	0,99
1,00	0,00	0,00	0,54	0,65

Таким чином, датасет оброблено та перетворено до формату, який можна використовувати для навчання.

4.2 Тренування та тестування моделей

Перед початком тренування моделі потрібно розділити оброблені дні на дві частини: групу безпосередньо для тренування (80 % від всього набору) та тестування (20 %). Перша буде використана в процесі навчання моделі – у якості дата сету для тренування, а друга – для визначення та порівняння отриманих результатів.

Процес тренування полягає у знаходженні значень для важелів W, а також векторів зсуву b, при яких цільова функція буде мати мінімальне можливе значення. Для знаходження таких значень зазвичай застосовують метод градієнтного спуску [7]. Ключова ідея цього методу полягає в тому, щоб зробити

повторні кроки у протилежному напрямку градієнта (або приблизного градієнта) функції в поточній точці, оскільки це є напрямком найкрутішого спуску. Градієнтний спуск у випадку важелів для двох елементів θ_0, θ_1 проілюстровано на рисунку 4.2.

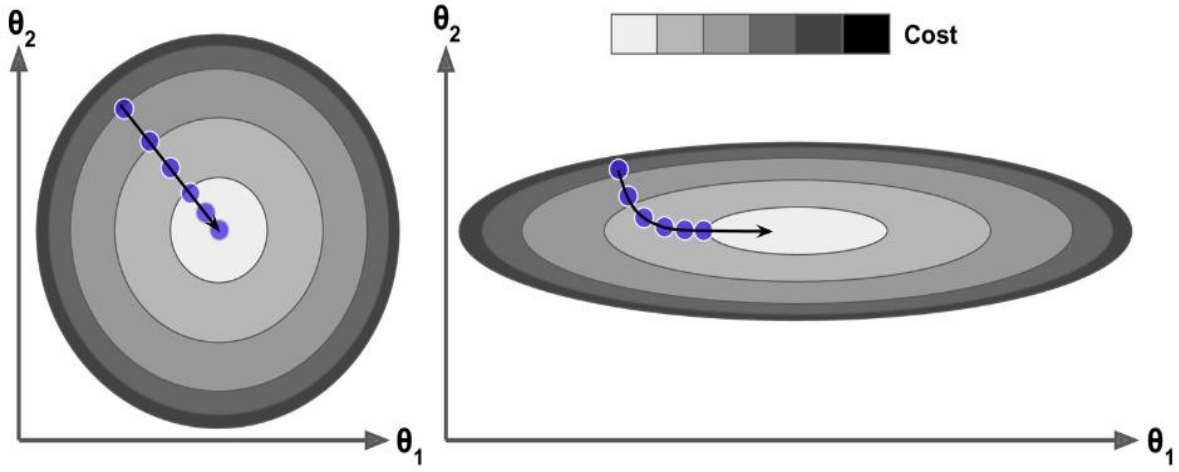


Рисунок 4.2 – Градієнтний спуск

Крім того, ефективність моделей залежить від гіперпараметрів – тобто набору налаштувань, які контролюють процес навчання та які можна змінювати для досягнення найкращого результату. Зазвичай такими можуть бути наступні параметри моделі: кількість епох навчання – кількість проходів через всі дані, розмір групи даних, який передається за один раз – розмір батчу, кількість клітин нейронної мережі.

Ще одним важливим гіперпараметром є коефіцієнт швидкості навчання, тобто число, на який помножується градієнт при переході на наступний етап градієнтного спуску. Якщо це значення є дуже великим, швидкість спуску буде більш високою, але він може знайти значно менш оптимальні важелі ніж тренування з маленьким коефіцієнтом. Але коли коефіцієнт швидкості є дуже малим час, який програма витратить для тренування моделі може виявитися досить значним.

Також для глибинних архітектур розглядається процент відсіву – гіперпараметр, який визначає вірогідність того, що деяка клітина буде відключена

на даному етапі тренування. Така техніка допомагає вирішувати проблему перенавчання, що виникає коли модель на тренувальних даних показує дуже високий результат, але у процесі тестування виявляється значно менш ефективною.

Модель навчалася на машині з наступними характеристиками: процесор – Intel Core i5 9300H, відеокарта Nvidia GTX 1660TI, об’єм оперативної пам’яті – 16 гігабайт. Тренування моделей було проведено за допомогою інструментів, які надає бібліотека для глибинного навчання tensorflow. Вона є одним з найпопулярніших та найбільш ефективних засобів тренування глибинних нейронних мереж. Крім того, ця бібліотека надає зручний високорівневий інтерфейс для роботи з шарами різних архітектур, зберігаючи при цьому гнучкість використання.

В якості гіперпараметрів було використано значення, які наведені у таблиці 4.2.

Таблиця 4.2 – Гіперпараметри нейронних мереж

	LSTM	GRU	LSTM-SVM	GRU-SVM
Розмір батчу	256	256	256	256
Кількість клітин	85	90	85	90
Процент відсіву	0.85	0.85	0.8	0.8
Коефіцієнт швидкості навчання	$1e^{-6}$	$1e^{-5}$	$1e^{-6}$	$1e^{-5}$
Кількість епох	5	5	6	6

Для порівняння результатів тестування натренованих моделей було використано наступні метрики: точність, процент вірних позитивні відповіді, процент невірних позитивних відповідей. Точність обчислюється за допомогою формули 4.2:

$$\text{точність} = \frac{\text{кількість правильних відповідей}}{\text{кількість тестових прикладів}} \times 100\% \qquad (4.2)$$

Процент вірних позитивних відповідей – відношення правильних позитивних результатів до загальної кількості випадків, які у даних позначено загрозою. Аналогічно процент невірних позитивних відповідей – відношення неправильних позитивних результатів до загальної кількості прикладів, які у даних позначено звичайними.

Результати тестування наведено в таблиці 4.3.

Таблиця 4.3 – Результати дослідження

	Точність (%)	Вірні позитивні відповіді (%)	Невірні позитивні відповіді (%)
LSTM	96,8	94,9	2,5
GRU	96,4	95,2	3.1
LSTM-SVM	96,1	94,7	2,7
GRU-SVM	96,2	95,1	2.1

Спираючись на отримані у ході перевірки натренованих моделей на тестовому наборі даних результати можна зробити висновок, що додавання SVM шару у якості класифікатора для рекурентної мережі не дає значного виграшу в ефективності на розглянутих даних. Точність LSTM рішення дещо вища за GRU, але остання має менший процент вірних позитивних відповідей, тому саме цей варіант моделі було обрано для реалізації мережевої системи виявлення вторгнень.

5 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Реалізована система є додатком для аналізу мережевого трафіку з окремих серверів кластеру на аномальну активність. Він складається з двох частин: агенту, який розгортується на окремих серверах та аналізатору, який обробляє інформацію, що була надіслана з агентів та відображає її на інтерфейсі користувача.

5.1 Опис агентської частини системи

Агент складається з двох окремих частин. Перша збирає дані мережевого трафіку за допомоги інтерфейсу ядра операційної системи та зберігає їх на диску в окремі файли у форматі pcap, кожен з яких займає приблизно 256 мегабайтів. Друга частина періодично проходить по списку збережених файлів, стискає їх та надсилає їх до серверу для подальшої обробки, а також видаляє ті файли, які були відправлені деякий час назад.

З точки зору операційної системи агенту перша частина є процесом-демоном, тобто працює в фоновому режимі. Це зроблено за допомогою обгортання Python програми, яка складається з модулю sniffer.py в systemd сервіс. Такі сервіси дозволяють ізолювати процеси в окремому середовищі та коректно запускати та відключати їх при завантажені та відключенні системи відповідно. Головним елементом програми є об’єкт класу Sniffer, що реалізує протокол контекстного менеджеру, який є досить поширеною технікою у Python. Це дозволяє йому коректно завершуватись та утілізовувати виділені ресурси при виникненні зовнішніх помилок.

Для збирання даних мережевого трафіку була використана бібліотека scapy, яка надає можливість взаємодіяти з інтерфейсом ядра операційної системи Linux

на дещо більш високому рівні. Фрагмент коду модулю sniffer.py наведено на рисунку 5.1.

```
class Sniffer:
    def __init__(self, iface):
        self.base_dir = Path(PCAP_DIR)
        self._sniffer = AsyncSniffer(
            iface=iface,
            prn=Sniffer.save_pcap,
            count=0
        )

    @classmethod
    def save_pcap(cls, pkt):
        ts = int(time.time())
        nonce = random.randint(0, MAX_NONCE)
        filename = f'pkt-{ts}-{nonce}.cap'
        wrpcap(filename, pkt)

    def __enter__(self):
        self.base_dir.mkdir(parents_ok=True, exist_ok=True)
        self._sniffer.start()

    def __exit__(self, exc_ty, exc_val, tb):
        self._sniffer.stop()
```

Рисунок 5.1 – Фрагмент коду sniffer.py

Друга частина, яка складається з модулю sender.py запускається періодично, тому в систему вона вбудована за допомогою сервісу cron. Він надає функції планувальника завдань, який дозволяє виконувати процеси періодично за деяким інтервалом або в чітко назначену дату та час. Безпосередньо програма переглядає всі файли з розширенням *.cap, які знаходяться у директорії, яку наведено за допомоги змінних середовища, та надсилає такі файли через проміжок часу, який також можна налаштувати. При цьому розглядаються тільки ті файли, які були створені після останньої відправки. Це реалізовано за допомогою допоміжного файлу, який зберігає час останнього завершення cron завдання. Крім того, всі файли, які було створено визначений проміжок часу назад, видаляються. Це

зробленого для того, щоб не забруднювати диск сервери такими даними. Фрагмент коду модулю sender.py наведено на рисунку 5.2.

```
def send_pcap(pcap_filepath):
    with open(pcap_filepath) as pcap_file:
        request.post(ENDPOINT, data=pcap_file)


def get_last_ts():
    with open(POS_FILEPATH) as pos_file:
        data = json.load(pos_file)
        return data['ts']


def save_last_ts():
    ts = time.time()
    with open(POS_FILEPATH, 'w') as pos_file:
        json.dump({'ts': ts}, pos_file)


def send_files():
    ts = get_last_ts()
    pcap_files = os.path.join(f'{BASE_DIR}', '*.cap')
    for filepath in glob.glob(pcap_files):
        if os.path.getmtime(filepath) > ts:
            send_pcap(filepath)
    save_last_ts()
```

Рисунок 5.2 – Фрагмент коду sniffer.py

Також слід зазначити, що перед відправкою всі pcap файли стискуються за допомогою бібліотеки gzip, що дозволяє зменшити розмір даних для передачі.

5.2 Опис роботи інтерфейсу користувача

Найбільш важливою функціональною частиною системи є сповіщення адміністраторів мережі. Зараз однією з найбільш популярних додатків для

корпоративного зв'язку є Slack. Це означає, що здебільшого інженери використовують його щодня та вміють налаштовувати повідомлення та їх пріоритет, а також інтеграції зі стороннім інтерфейсом. Тому є доцільним відправляти нотифікації про зловмисні дії, використовуючи Slack.

Система має два типи повідомлень. По-перше, повідомлення про загрози, яке надсилається, коли підозріла дія була знайдена, а також містить IP-адресу серверу, на якому виявлено зловмисну активність. Приклад такого типу нотифікації наведено на рисунку 5.3:

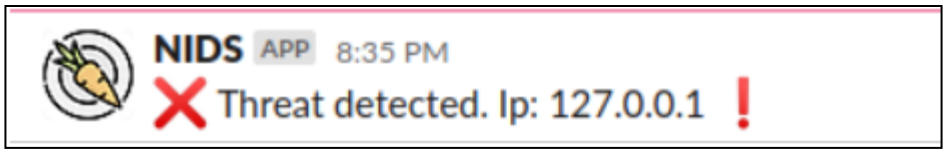


Рисунок 5.3 – Нотифікація про загрозу

По-друге, повідомлення про статус системи, яке надсилається періодично за допомогою cron завдань. Тобто у випадку, коли робота додатку порушена (він не відповідає на запити), буде надіслано повідомлення про проблеми або нестабільну роботу системи. Якщо додаток працює у нормальному режимі користувачі отримають нотифікацію зі статусом «ок» два рази на день. Приклад такого повідомлення наведено на рисунку 5.4.

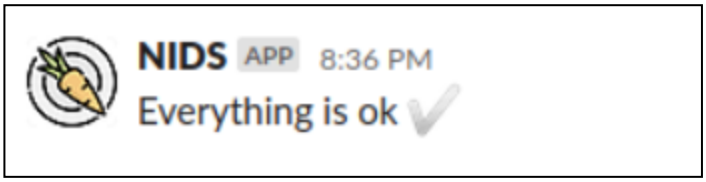


Рисунок 5.4 – Нотифікація про статус системи

Ще одним важливим елементом системи є інтерфейс для візуалізації інцидентів, який дозволяє моніторити інформацію з трафіку, відображати її за допомогою діаграм та графіків і, таким чином, приймати додаткові рішення на їх основі. Цей інтерфейс запускається автоматично при встановлені аналізатору.

Щоб отримати доступ до нього потрібно авторизуватися за допомогою форми, що зображено на рисунку 5.5.

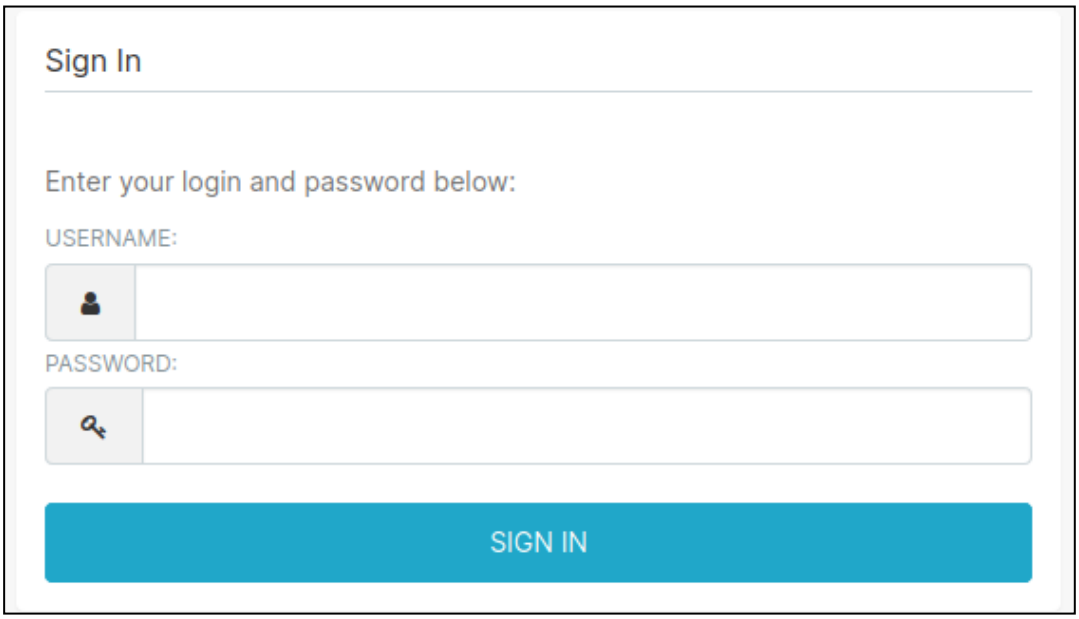
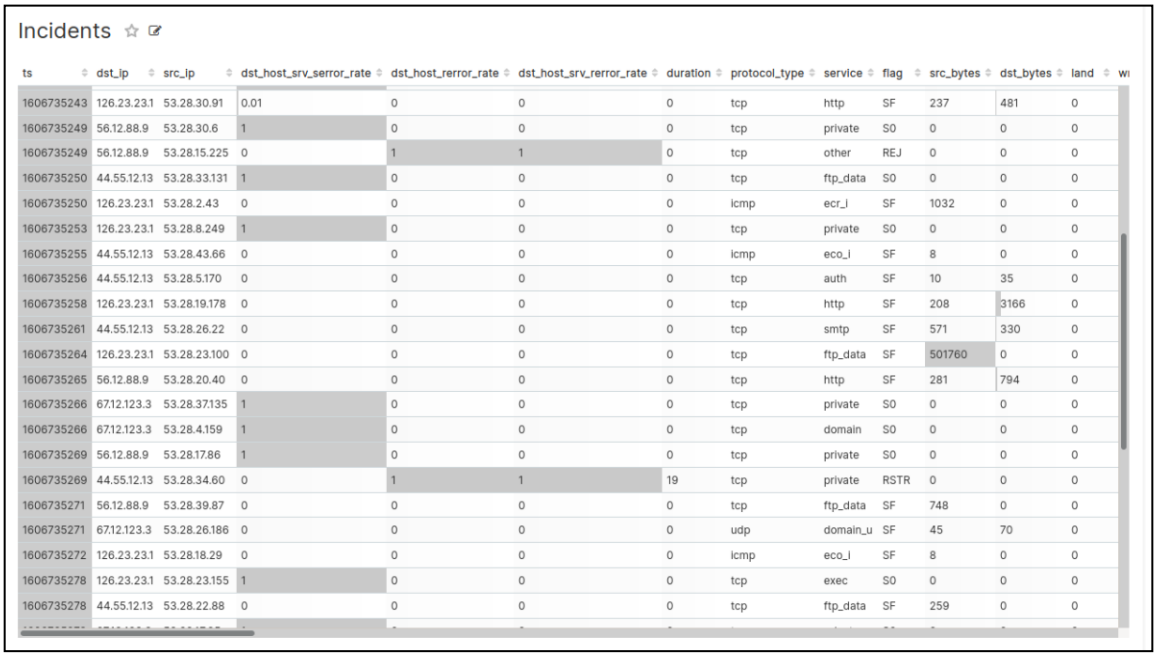


Рисунок 5.5 – Вхід у систему

Після цього користувач опиняється на головній сторінці додатку, де наведена таблиця з інцидентами, з інформацією по кожному з них. Загальний вигляд цієї частини інтерфейсу наведено на рисунку 5.6.


Incidents

ts	dst_ip	src_ip	dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wr
1606735243	126.23.23.1	53.28.30.91	0.01	0	0	0	tcp	http	SF	237	481	0	
1606735249	56.12.88.9	53.28.30.6	1	0	0	0	tcp	private	S0	0	0	0	
1606735249	56.12.88.9	53.28.15.225	0	1	1	0	tcp	other	REJ	0	0	0	
1606735250	44.55.12.13	53.28.33.131	1	0	0	0	tcp	ftp_data	S0	0	0	0	
1606735250	126.23.23.1	53.28.2.43	0	0	0	0	icmp	ecr_i	SF	1032	0	0	
1606735253	126.23.23.1	53.28.8.249	1	0	0	0	tcp	private	S0	0	0	0	
1606735255	44.55.12.13	53.28.43.66	0	0	0	0	icmp	eco_i	SF	8	0	0	
1606735256	44.55.12.13	53.28.5.170	0	0	0	0	tcp	auth	SF	10	35	0	
1606735258	126.23.23.1	53.28.19.178	0	0	0	0	tcp	http	SF	208	3166	0	
1606735261	44.55.12.13	53.28.26.22	0	0	0	0	tcp	smtp	SF	571	330	0	
1606735264	126.23.23.1	53.28.23.100	0	0	0	0	tcp	ftp_data	SF	501760	0	0	
1606735265	56.12.88.9	53.28.20.40	0	0	0	0	tcp	http	SF	281	794	0	
1606735266	67.12.123.3	53.28.37.135	1	0	0	0	tcp	private	S0	0	0	0	
1606735266	67.12.123.3	53.28.4.159	1	0	0	0	tcp	domain	S0	0	0	0	
1606735269	56.12.88.9	53.28.17.86	1	0	0	0	tcp	private	S0	0	0	0	
1606735269	44.55.12.13	53.28.34.60	0	1	1	19	tcp	private	RSTR	0	0	0	
1606735271	56.12.88.9	53.28.39.87	0	0	0	0	tcp	ftp_data	SF	748	0	0	
1606735271	67.12.123.3	53.28.26.186	0	0	0	0	udp	domain_u	SF	45	70	0	
1606735272	126.23.23.1	53.28.18.29	0	0	0	0	icmp	eco_i	SF	8	0	0	
1606735278	126.23.23.1	53.28.23.155	1	0	0	0	tcp	exec	S0	0	0	0	
1606735278	44.55.12.13	53.28.22.88	0	0	0	0	tcp	ftp_data	SF	259	0	0	

Рисунок 5.6 – Таблиця з інцидентами

Таблиця надає можливість обирати стовпці для відображення та обмежувати кількість строк. Також надана можливість фільтрувати дані за значенням параметрів. Інтерфейс цього функціоналу проілюстровано на рисунку 5.7.

Рисунок 5.7 – Фільтрація інцидентів

Крім того, є можливість сортувати елементи таблиці за одним з параметрів, а також обирати порядок – за зростанням та спаданням. Панель сортування наведено на рисунку 5.8.

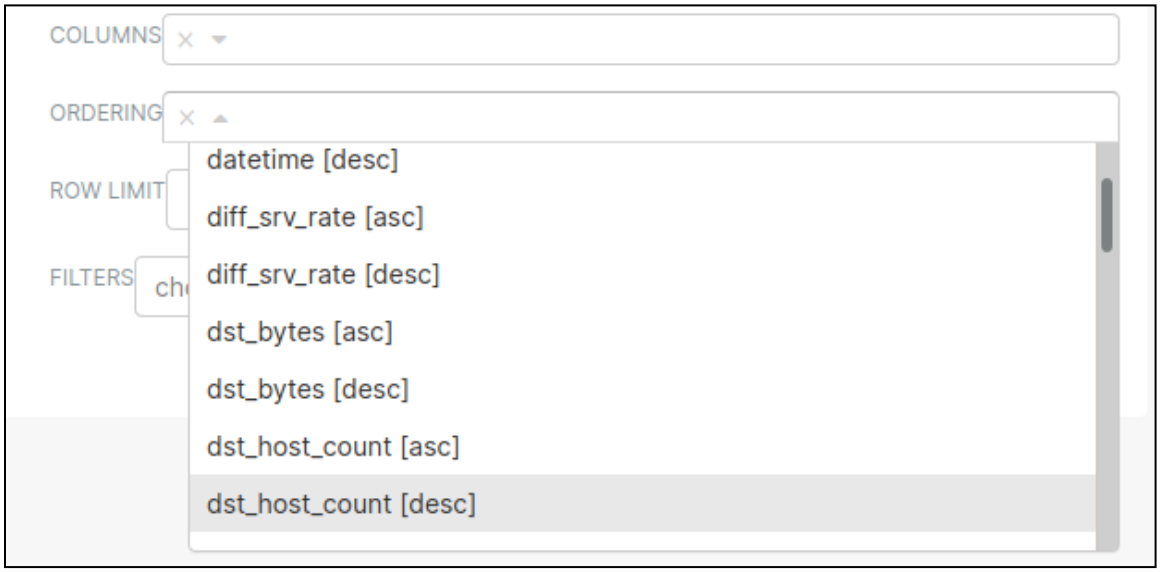


Рисунок 5.8 – Сортування інцидентів

Система надає функціонал візуалізації дані по інцидентів за допомогою діаграм та графіків, що може допомогти проаналізувати виявленні інциденти та визначити, які компоненти потребують додаткової уваги. На рисунку 5.9 наведено приклад такої візуалізації: залежність інцидентів від часу та маркування зображення загрозливих інцидентів.

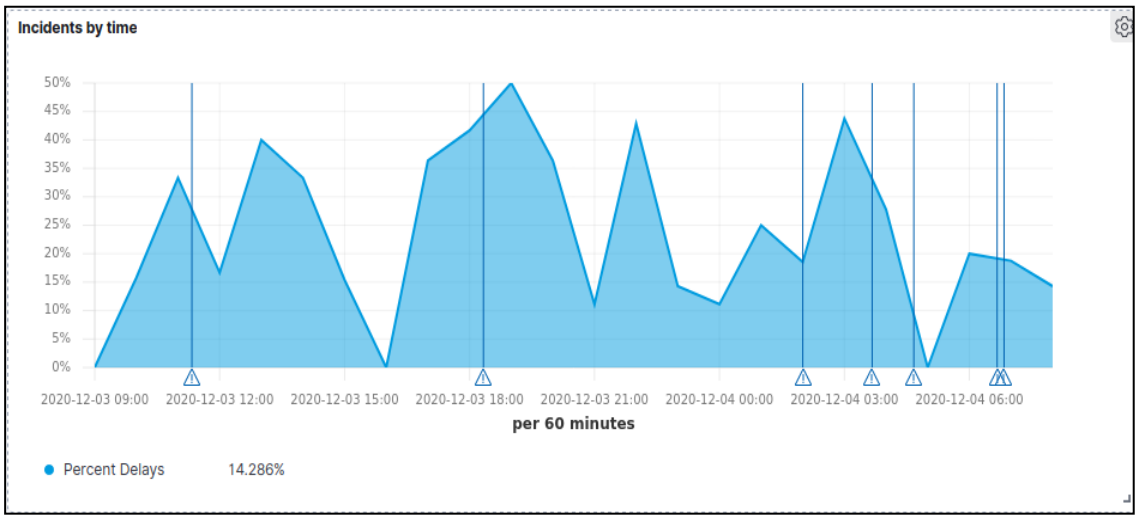


Рисунок 5.9 – Графік інцидентів за часом

Також є можливість відобразити стовпчасті діаграми кількості інцидентів з визначеної адреси. Приклад такої візуалізації наведено на малюнку 5.10.

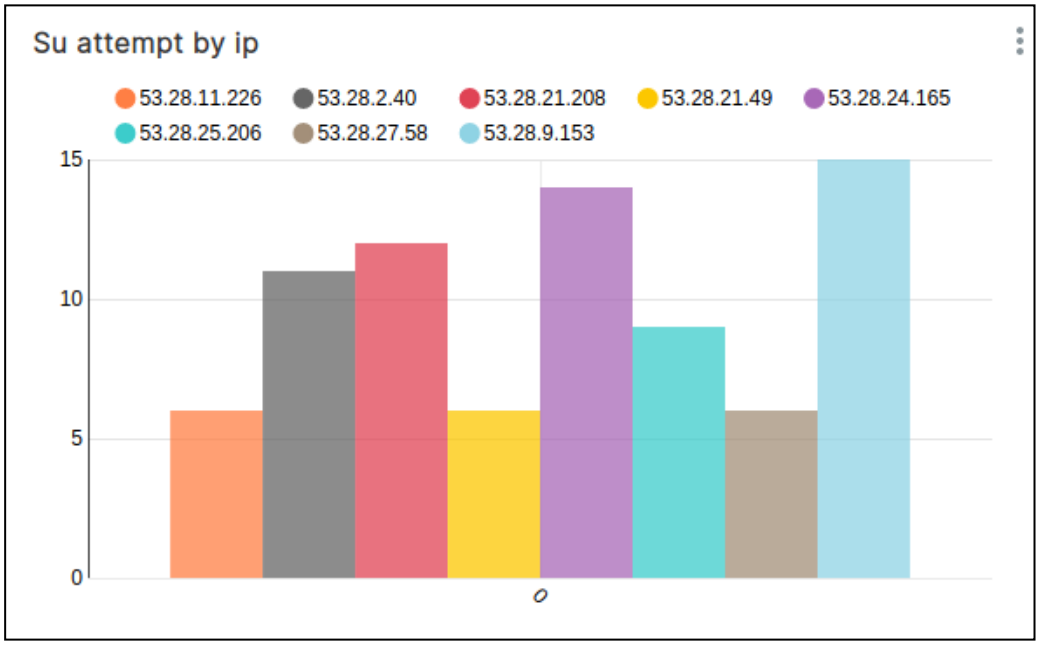


Рисунок 5.10 – Діаграма інцидентів за IP-адресою

Досить виразними с точки зору користувача є так звані «pie-charts» – діаграми у вигляді кіл, що розділені на сектори, кожен з яких позначає окремий кластер даних. Система дає можливість будувати візуалізації такого відображення, а саме продемонструвати кількість інцидентів за кожним з протоколів, що зображено на рисунку 5.11.

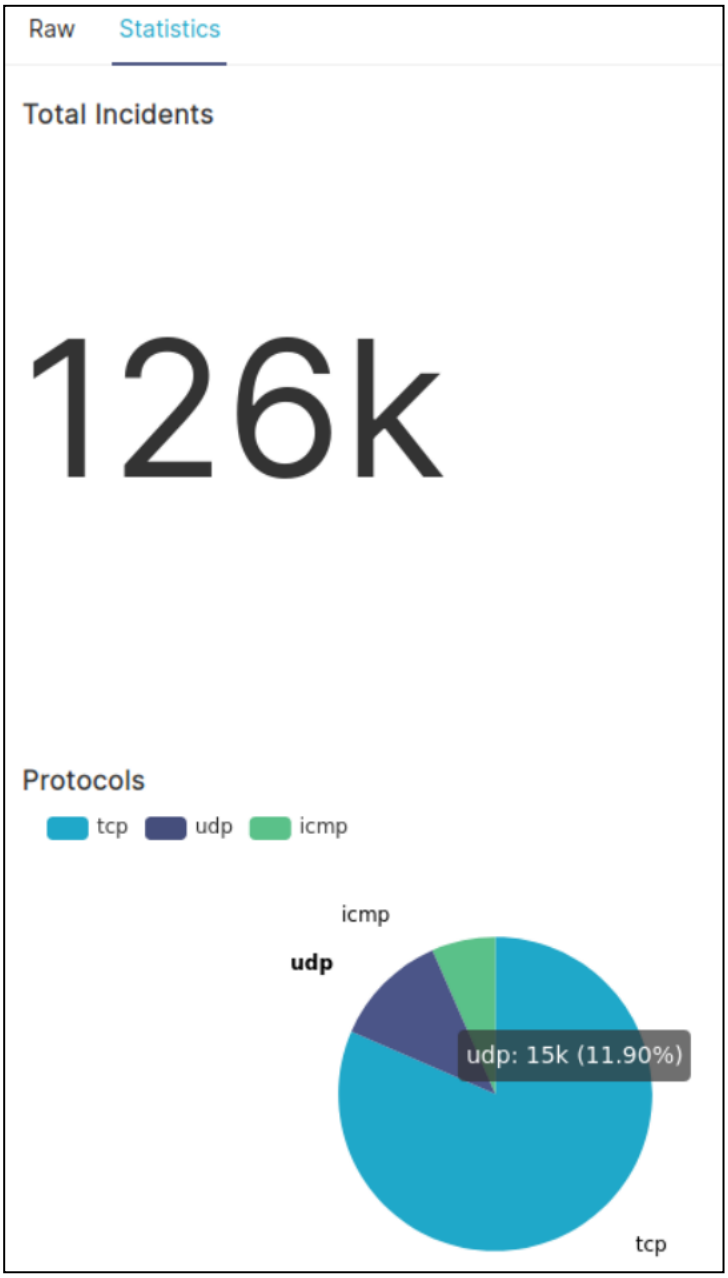


Рисунок 5.11 – Діаграма інцидентів за протоколами

Для перегляду даних про користувача потрібно натиснути на кнопку «Settings» у правому верхньому куті та обрати секцію «User». Після цього

відкриється вікно з формою, яке зображено на рисунку 5.12, де наведено всю інформацію.

Рисунок 5.12 – Інформація про користувача

Email – статична інформація, яка є унікальним ідентифікатором користувача. Але деякі дані можна змінити. Наприклад ім'я та прізвище. Також доступна опція зміну паролю, яка відображена на рисунку 5.13.

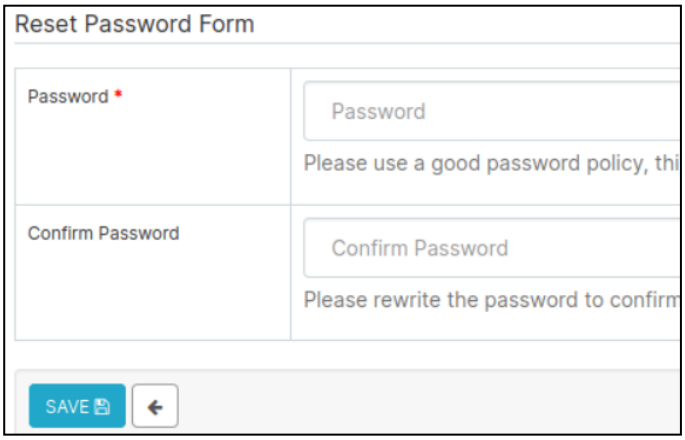


Рисунок 5.13 – Зміна паролю

Для отримання доступу до мережі користувач повинен бути доданий одним з адміністраторів серверу. Це обумовлено тим, що кількість людей, що буде використовувати систему досить обмежена.

5.3 Опис процесу розгортання системи

Як було зазначено на етапі визначення вимог, система повинна надавати можливість швидко розгортати її на декілька серверів одразу. Для цього був використаний ansible – система для автоматизації налаштування, який використовує визначену структуру файлів та директорій, кожна з яких повинна бути позначена належним чином. Одним із таких елементів є ролі, кожна з яких виконує визначену задачу, наприклад, встановлення системних пакетів. Ролі, для розробленої системи наведено на рисунку 5.14.

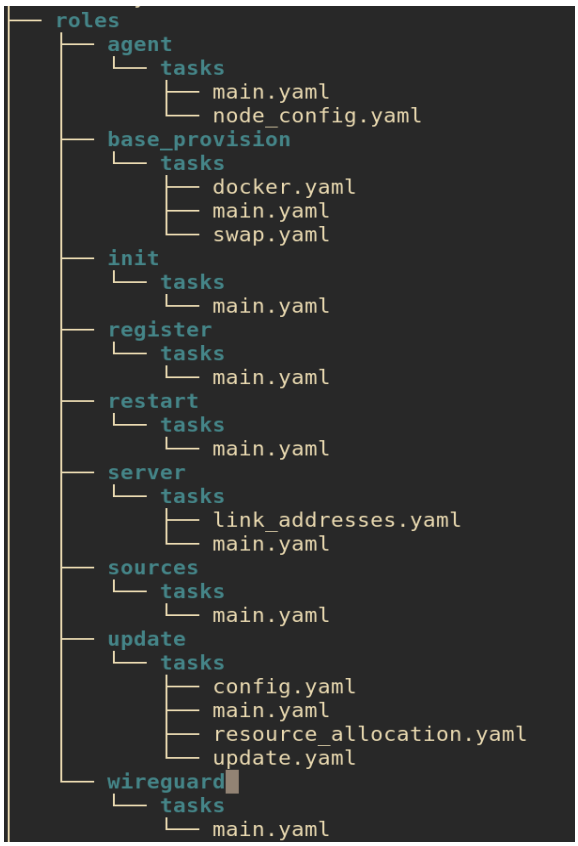


Рисунок 5.14 – Ролі ansible

Ansible – один з найпопулярніших інструментів системних інженерів та розробників, тому вони зможуть з допомоги однієї команди в терміналі встановити систему на свої сервери. Крім того, конфігураційні файли ansible можна інтегрувати з іншими інструментами, що робить рішення досить універсальним.

Для розгортання безпосередньо додатків, які працюють на моніторинговому сервері було використано docker – систему контейнеризації, яка дозволяє ізолювати процеси один від одного, підтримуючи при цьому деякі можливості операційної системи для кожного з них. Крім того, даний інструмент дозволяє переносити і клонувати контейнери, що сформовані для даних процесів, на інші машини. Частину конфігураційного файлу docker наведено на рисунку 5.15.

```
  rabbitmq
    image: rabbitmq:3.8
    container_name: rabbitmq
    restart: unless-stopped
    ports:
      - "127.0.0.1:15671:15671"
    volumes:
      - rabbitmq:/data

  db:
    env_file: docker/.env
    image: postgres:10
    container_name: postgres
    restart: unless-stopped
    ports:
      - "127.0.0.1:5432:5432"
    volumes:
      - db_home:/var/lib/postgresql/data

  client-api:
    env_file: docker/.env
    image: *client-api
    container_name: client_api
    command: ["/app/docker/client_api.sh"]
    restart: unless-stopped
    ports:
      - 8088:8088
    depends_on: *superset-depends-on
    volumes: *superset-volumes
```

Рисунок 5.15 – Конфігурація додатків серверу

Як було зазначено вище, для розгортання агентської частини один з двох процесів було обгорнуто в systemd сервіс. Це зроблено за допомогою додавання конфігураційного файлу, наведеного на рисунку 5.16.

```
[Unit]
Description=python agent
Conflicts=getty@tty1.service
After=network.target

[Service]
Type=simple
WorkingDirectory=/opt/nids-agent/
ExecStart=/opt/nids-agent/venv/bin/python3 app.py
EnvironmentFile=/etc/nids-agent/env
Restart=always
KillSignal=SIGINT
StandardError=syslog
NotifyAccess=all

[Install]
WantedBy=multi-user.target
```

Рисунок 5.16 – Конфігурація агентського сервісу

В цьому файлі визначаються наступні параметри: мета інформацію процесу, робочу директорію, ім’я файлу, що потрібно виконати, правила завантаження, а також місце, куди потрібно записувати файли журналів.

Таким чином, була описана програмна реалізація мережевої системи виявлення вторгнень, яка включає агентську частину, моніторинговий сервер, а також конфігураційні файли для розгортання на машини користувачів.

6 АНАЛІЗ ДОСЛІДНИЦЬКОЇ ЕКСПЛУАТАЦІЇ ТА МОЖЛИВИХ ЗАСТОСУВАНЬ

6.1 Аналіз можливих застосувань

В ході виконання магістерської атестаційної роботи було досліджено методи глибинного машинного навчання, а також спроектовано та розроблено мережеву систему виявлення вторгнень, яка дає можливість використати отриманні моделі для визначення загроз до власних хмарних серверів. Натреновані моделі можна використати для будування інших додатків моніторингу трафіку, або ж розширення функціоналу та ефективності існуючих шляхом додавання відповідного сервісу. Програмну систему, що розроблено можна використати, як аналізатор подій у мережі та систему сповіщення. Також інтерфейс користувача є досить функціональним інструментом для перегляду статистики та візуалізації інцидентів, а також дослідження загроз у подальшій роботі.

Розроблений додаток призначений перш за все для інженерів та адміністраторів хмарних серверів, які зацікавленні у збільшені рівня безпеки власних ресурсів або додавання рішення для нотифікації про загрози, впевнено користуються популярними інструментами для автоматизації налаштування та розгортання, а також активно використовують додаток Slack для комунікації всередині команди.

У подальшому можна адаптувати натреновану моделі на предмет класифікації загроз за типами та групами. До розробленої програмної системи можна додати функціонал сповіщення користувачів за допомогою електроної пошти, мобільного телефону та інших каналів комунікації. Крім того, доцільним буде додавання інтерфейсів для сторонніх програм візуалізації таких як Grafana або Kibana, що розширити можливості користувачів системи та допоможуть їм адаптувати власні розробки та додатки під мережеву систему виявлення вторгнень.

6.2 Опис тестування системи

У загальному сенсі тестування програмного забезпечення – перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином та в більшості випадків автоматизовано за допомогою спеціальних засобів, яки надають різноманітні технології. У процесі розробки мережевої системи виявлення вторгнень використовувались модульне та функціональне тестування, а також тестування навантаження.

Модульне тестування – процес у програмуванні, що дозволяє перевірити на коректність окремі модулі вихідного коду програми. Ідея полягає в тому, щоб писати тести для кожної нетривіальною функції або методу. Це дозволяє досить швидко перевірити, чи не призвела чергова зміна коду до регресії, тобто до появи помилок в уже протестованих місцях програми, а також полегшує виявлення і усунення таких помилок, а також значно зменшити час, який буде відведено на кінцеве тестування та налагодження програми.

Для написання модульних тестів був використаний Python-інструмент для тестування pytest. Він вбудований до стандартної бібліотеки та не вимагає додаткових завантажень. Головна перевага pytest полягає в можливості виконувати автоматичне тестування без необхідності писати одні і ті ж тести багато разів і запам'ятовувати правильні результати їх виконання.

Також для тестування використовувалися mock-об’єкти за допомогою бібліотеки mock. Mock-об’єкт являє собою конкретну фіктивну реалізацію інтерфейсу, призначену виключно для тестування взаємодії і щодо якого висловлюється твердження. Його доцільно використовувати, коли треба провести тестування функції, яка використовує зовнішні елементи, наприклад запити до зовнішнього API або складними та складними, ресурсомісткими бібліотеками чи іншими елементами, протестувати які в режимі модульного тестування є складним завданням.

Функціональне тестування фактично є аналізом характеристик додатку, а також перевірку на відповідність між реальною поведінкою реалізованих функцій та очікуваною, яку визначають бізнес-вимоги. Функціональне тестування фактично емулює використання системи. в ході роботи його було було здійснено за допомогою ручного тестування системи, шляхом моделювання загроз, використовуючи інструмент nmap, на декілька серверів, які було розгорнуто на DigitalOcean та на які було розгорнуто розроблену систему.

Тестування навантаження – перевірка продуктивності, збір показників та визначення часу відгуку програмної системи у відповідь на зовнішній запит з метою встановлення відповідності з вимогами. У рамках роботи було проведено стрес-тестування, при якому навантаження, що створюється на систему перевищувало нормальні сценарії використання. У ході серії експериментів було виявлено, що система здатна коректно обробляти запити при підключенні семи агентів.

В ході тестування було виявлено ряд незначних помилок, які успішно виправлені в подальшому. Завдяки вживанню таких заходів рівень якості програмної системи було доведено до рівня, при якому її можна впроваджувати кінцевим користувачам.

ВИСНОВКИ

В ході даної роботи було проведено дослідження методів глибинного машинного навчання для вирішення задачі побудови мережевої системи виявлення вторгнень.

Зокрема була проаналізована проблемна область інформаційної безпеки хмарних ресурсів, виявлені основні типи загроз, а також методи боротьби з ними, визначено призначення систем виявлення вторгнень та їх різновиди. Також були проаналізовані існуючі аналоги – були виявлені переваги та недоліки таких систем виявлення вторгнень, як Snort, Zeek та Suricata.

Крім того, були визначені вимоги до програмного продукту та призначення розробки.

Також проведено аналіз та UML-моделювання предметної області проаналізовані основні елементи системи виявлення вторгнень та ключові функції для різних користувачів. Визначені актори системи та основні варіанти використання та побудована відповідна UML-діаграма.

Проведений аналіз основних методів машинного навчання та обрані два основні, які було досліджено більш змістовно. Також запропоновано альтернативний метод для вирішення задачі виявлення зловмисного трафіку.

Крім того, обрано датасет, який потім було підготовлено та оброблено для навчання та проведено тренування моделі, яку потім було перевірено на тестових даних. За результатами дослідження було обрано рішення для наступного впровадження.

Також спроектована база даних. Були визначені основні сутності та їх атрибути та зв’язки. Крім того, була побудована схема бази даних.

В роботі розроблено архітектура системи виявлення вторгнень на основі методів глибинного машинного навчання для серверів, які працюють на операційних системах, заснованих на ядрі Linux: визначені основні компоненти, їх структурні елементи та процес взаємодії в різних випадках. Для опису загальної

архітектури системи була побудована діаграма розгортання. Також основні компоненти додатку описані за допомогою відповідної діаграми.

У подальшому було розроблено мережеву систему виявлення вторгнень, яка складається з агентської частини та серверу, який виконує функцію обробки зібраних даних, а також підготовлені конфігураційні файли для розгортання.

Натреновану модель можна використати для побудування моніторингових систем або розширення функціоналу вже існуючих. Розроблену систему може бути використано для моніторингу мережі різних хмарних серверів, кластерів, або локальних мереж та виявлення різноманітних загроз.

За результатами роботи були опубліковані тези та зроблена доповідь на Молодіжному форумі (див. додаток В) та конференції «Всеукраїнська науково-практична конференція. Комп’ютерна інженерія і кібербезпека: досягнення та інновації» 25-27 листопада 2020 року (див. додаток Г).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бобало Ю.Я., Горбатий І.В., Кіселичник М.Д., Бондарєв А.П. Інформаційна безпека. – Львів: Львівська політехніка, 2019. – 580 с.

2. Kachko O., Bilous N, Semerkov V. Research on methods for secure web applications development // Information Technologies in Innovation Business (ITIB). Proceedings of ITIB, Kharkiv, Ukraine. – 7-9 October 2015. С. 24-27.

3. Douligeris, Christos; Serpanos, Dimitrios N. Network Security: Current Status and Thoughts about Future Directions. New York City: John Wiley & Sons, 2007. – 453 с.

4. Vilela, Douglas ARTMAP Neural Network IDS Evaluation solutions applied for real world problems// International Joint Conference on Neural Networks (IJCNN). 2018. С. 7.

5. NIDS Guide plus best NIDS Software & Tools // Comparitech: Thousands of hours of in-depth tech research. Дата оновлення: 12.08.2020. URL: https://www.comparitech.com/net-admin/nids-tools-software/ (дата звернення: 04.09.2020).

6. Єрохін А., Пашкевич Є. Исследование методов и средств мониторинга серверов // Системи обробки інформації. 2011. №4. С. 188-192.

7. L. P. Dias, J. J. F. Cerqueira, K. D. R., Almeida R. Using artificial neural network in intrusion detection systems to computer networks // 9th Computer Science and Electronic Engineering (CEEC). 2017. С 10.

8. What is a DoS Attack? // Cloudflare official learning center website. Дата оновлення: 08.08.2019. URL: https://www.cloudflare.com/learning/ddos/what-is-a-ddos-attack (дата звернення: 14.09.2020).

9. What is SQL injection and how can you prevent it from happening // Penta Security Systems official blog website. Дата оновлення: 02.07.2020. URL: https://www.pentasecurity.com/blog/sql-injection-vulnerabilities (дата звернення: 15.09.2020).

10. Lennon Y., C. Chang. Cybercrime and establishing a secure cyberworld // New York: Springer. 2014. С. 49-64.

11. Mafra P.M., Fraga J.S., Santin A.O. Algorithms for a distributed IDS in MANET // Journal of Computer and System Sciences. 2014. С. 554-570.

12. John R. Vacca Network and System Security. – Amsterdam: Syngress Publishing. 2014. – 432 с.

13. Maksym Bekuzarov, Mariya Shirokopetleva, Oleksandr Samantsov, Oksana Mazurova Neural Network Architecture Editor With Code Generation // Problem of Infocommunications. Science and Technolpgy (PIC S&T′2020), Kharkiv, Ukraine. – 6-9 October 2020 (Scopus). С. 35-39.

14. Мазурова О., Широкопетлева М. Поддержка Проектирования Баз Данных // Інформаційні системи та технології: матеріали міжнар. наук.-техн. конф, Харків, Україна. – 10-15 вересня 2018. С. 319-322.

15. Страхарчук А. Я. Інформаційні системи і технології. – Київ: УБС НБУ: Знання, 2010. – 515 с.

16. Snort – one of the best intrusion detection system // Snort project official website. Дата оновлення: 03.09.2020. URL: https://www.snort.org/ (дата звернення: 24.09.2020).

17. An Open Source Network Security Monitoring Tool // Zeek project official website. Дата оновлення: 14.06.2020. URL: https://zeek.org/ (дата звернення: 25.09.2020).

18. Suricata intrusion detection system // Suricata project official web site. Дата оновлення: 15.05.2020. URL: https://suricata-ids.org (дата звернення: 25.09.2020).

19. Libcap man page // Linux man website. Дата оновлення: 25.04.2020. URL: https://www.man7.org/linux/man-pages/man3/libcap.3.html (дата звернення: 27.09.2020).

20. Буч Г. Язык UML. Руководство пользователя. – Москва: ДМК Пресс, 2006. – 496 с.

21. L. Deng, D. Yu. Deep Learning: Methods and Applications // Foundations and Trends in Signal Processing. 2014. С. 1-19.

22. Zell A, Andreas P. Simulation of Neural Network. – Boston: Addison-Wesley. 2003. – 456 с.

23. Andrey Arsenov, Igor Ruban, Kyrylo Smelyakov, Anastasiya Chupryna . Evolution of Convolutional Neural Network Architecture in Image Classification Problems // Selected Papers of the XVIII International Scientific and Practical Conference on IT and Security (ITS). Kiyv, Ukraine. 2019. С. 9-10.

24. Oppel, Andy. Databases Demystified. — San Francisco: McGraw-Hill Osborne Media, 2015. – 150 с.

25. Генельт А.Е. Автоматизированные методы разработки архитектуры программного обеспечения // Сайт Санкт-Петербургского государственного университета информационных технологий, механики и оптики. Дата оновлення: 19.07.2020. URL: http://is.ifmo.ru/books/_henelt2.pdf. (дата звернення: 28.09.2020).

26. Inturstion datasets // Canadian Institute for Cybersecurity official web site. Дата оновлення: 11.03.2020. URL: https://www.unb.ca/cic/datasets/ (дата звернення: 29.10.2020).