Міністерство освіти і науки України
Національний авіаційний університет
Кафедра комп'ютеризованих систем управління

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Литвиненко О.Є.
"_____"_____ 2022 р.

ДИПЛОМНИЙ ПРОЄКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)
ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: Підсистема підтримки прийняття рішень при виникненні пожежі в салоні повітряного судна _____

Виконавець: _____ Горбань Ю.О.

Керівник: _____ доц., к.ф.м.-н., Кучерява О.М.

Нормоконтролер: _____ Кучерява О.М.

Київ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 126 «Інформаційні системи та технології»

(шифр, найменування)

Освітньо-професійна програма «Інформаційні системи та технології»

Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри
Литвиненко О.Є.
«_____»_____2022 р.

ЗАВДАННЯ

на виконання дипломного проєкту

Тенягіна Дімитрія Дмитровича

(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломного проєкту: Підсистема підтримки прийняття рішень при виникненні пожежі в салоні повітряного судна

затверджена наказом ректора від 15.02.2022 р. № 251/ст.

2. Термін виконання проекту: з 16.05.2022 р. по 19.06.2022 р

3. Вихідні дані до проєкту: мова програмування *C#*, середовище розробки *Microsoft Visual Studio*, декларативна мова розмітки *XAML*, платформа *.NET Framework*, платформа *Universal Windows Platform*.

4. Зміст пояснювальної записки:

1) Теоретичні аспекти реалізації підсистеми підтримки прийняття рішень
2) Технології для розробки підсистеми підтримки прийняття рішень
3) Розробка та демонстрація підсистеми підтримки прийняття рішень

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Ілюстрація графічного інтерфейсу програми
2) Схема алгоритму методу *Timer_Tick()*
3) Схема алгоритму конструктора *Airport()*

4) Схема алгоритму методу *Change_Recom()*

Календарний план-графік

/п	Етапи виконання дипломного проєкту	Термін виконання етапів	Від мітка про виконання
1.	Ознайомитись з матеріалами за темою проєкту. Підготувати текст першого розділу пояснювальної записки.	16.05.2022 – 20.05.2022	
2.	Визначити функціонал та архітектуру підсистеми підтримки прийняття рішень.	21.05.2022 – 25.05.2022	
3.	Проаналізувати методи та технології розробки підсистеми підтримки прийняття рішень. Підготувати текст другого розділу пояснювальної записки.	26.05.2022 – 01.06.2022	
4.	Розробка алгоритму, написання та компіляція програмного коду. Підготувати текст третього розділу.	02.06.2022 – 05.06.2022	
5.	Завершити оформлення пояснювальної записки. Оформити графічний матеріал.	06.06.2022 – 10.06.2022	
6.	Підготувати доповідь та презентацію.	15.06.2022 – 16.06.2022	
7.	Захист дипломного проєкту	18.06.2022	

7. Дата видачі завдання: «16» травня 2022 р.

Керівник дипломного проекту _____ Кучерява О.М.
(підпис керівника)

Завдання прийняв до виконання _____ Горбань Ю.О.

ЗМІСТ

ВСТУП

Стрімке впровадження інфокомунікаційних технологій у різні сфери людської діяльності сприяє розвитку методів протидії їхньому функціонуванню для стримування пропонованих ними можливостей із боку зловмисників. Причому така протидія реалізується на різних рівнях для впливу як на приватних осіб, так і державні структури. Внаслідок цього з'являється безліч класів деструктивних інформаційних кібернетичних впливів (ДІКВ).

Загальний аналіз атак показує, що одним із ефективних способів впливів на інфокомунікаційну мережу з метою порушення процесів управління нею є несанкціоноване блокування доступу до інформаційних ресурсів. Найбільш поширеним методом ДІКВ є розподілена атака відмови в обслуговуванні (DDoS-атака).

Результати аналізу ДІКВ на інфокомунікаційні мережі у 2021 році вказують на те, що однією з основних цілей ДІКВ є прикладний рівень інфокомунікаційної мережі. Статистика атак за кількістю запитів на секунду показує, що більш ніж у 79,4% випадків атак потужність атаки лежить у межах 10-1000 тис. запитів на секунду (RPS) при середній тривалості від 30 до 60 хвилин на 41,4% випадків.

Розглядаючи динаміку тенденції проведення розподілених атак відмови в обслуговуванні прикладного рівня, можна дати прогноз збільшення в найближчому майбутньому частки застосування низькоінтенсивних атак, що потребує вдосконалення методів їх виявлення.

З метою мінімізації наслідків DDoS-атак, їх виявлення та класифікація є вкрай важливим і водночас складним завданням. Основний спосіб розпізнавання DDoS-атаки полягає у виявленні аномалій у структурі трафіку. Традиційні механізми забезпечення безпеки – міжмережеві екрани та системи виявлення вторгнень – не є ефективними засобами для виявлення DDoS-атак та захисту від них, особливо атак трафіком великого обсягу.

Фундаментальною передумовою виявлення атак є побудова контрольних характеристик трафіку під час роботи мережі у штатних умовах із наступним пошуком аномалій у структурі трафіку (відхилення від контрольних характеристик).

Існуючі методи виявлення DoS-атак, засновані на статистичному аналізі порогових значень, що дозволяють ефективно розпізнавати атаки транспортного рівня (SYN-флуд, UDP-флуд та інші), малоефективні для виявлення низькоінтенсивних DoS-атак (Low-Rate DoS) прикладного рівня. Це зумовлює актуальність розробки нових механізмів виявлення низькоінтенсивних атак прикладного рівня типу «відмова в обслуговуванні» в комп'ютерних мережах за допомогою методів штучного інтелекту.

Отже метою роботи є розробка технології виявлення мережевих атак низької ефективності на основі аналізу трафіку по повному спектру параметрів з використання методів штучного інтелекту.

1. ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Низькоінтенсивні мережеві атаки

У сучасних мережах зв'язку найбільш поширеним і ефективним типом руйнуючого інформаційного впливу є атака «відмова в обслуговуванні» [1; 2]. Найбільш поширена схема реалізації подібних атак - задіяти ресурси атакуваного сервера виконанням фальшивих запитів на обслуговування, для створення яких несанкціоновано використовуються кілька серверів, розташованих у різних ділянках глобальної мережі. В результаті легальні користувачі отримують відмову в обслуговуванні, внаслідок чого дані руйнівні дії отримали назву DDoS (Distrib-uted Denial of Service). Жертвами подібних атак ставали такі відомі Web-портали, як Yahoo!, eBay, CNN.com, Amazon.com [3]. Сучасні системи захисту, що обслуговуються висококваліфікованими спеціалістами з мережевої безпеки, ставали безпорадними перед DDoS атаками, і в результаті сервери компаній повністю вийшли з ладу на тривалий час. Ефективних та універсальних засобів захисту від DDoS в даний час не існує. Крім того, регулярно з'являються все більш витончені атаки DDoS, що використовують нові можливості для руйнівних впливів. Отже, розвиток методів протидії конкретним типам DDoS атак є найбільш перспективним напрямом підвищення ефективності засобів забезпечення мережі безпеки.

Близько 90% DDoS атак використовують вразливість протоколу TCP [4]. Широко відомим прикладом подібного роду є атака, відома як затоплення черги напіввідкритих з'єднань сервера, що атакується пакетами із запитами на встановлення з'єднання (TCP SYN flooding). Дослідження атак даного виду широко представлені у науково-технічній літературі, розроблено безліч методів захисту від TCP SYN flooding. Проте питання їхньої якості залишається відкритим. Деякі схеми захисту, такі як SYN cookie та SYN cache, були рекомендовані Службою реагування на комп'ютерні інциденти

CERT (Computer Emergency Rresponse Team) [5], але не підтримані іншими експертами, які вказують на небажаність модифікації протоколу TCP/IP. Фахівці Європейського інституту мережі безпеки наголошують на зміцненні стека TCP/IP, відповідний інструментарій реалізований, наприклад, в операційних системах: Microsoft Windows 2000, RedHat Linux 7.3, Sun Solaris 8 і HP-UX 11.00. Останнім часом з'явилося кілька нових механізмів захисту від SYN flooding, заснованих на фільтрації пакетів, що відрізняються методами отримання інформації для аналізу трафіку і критеріями виявлення фальшивих пакетів. Незважаючи на те, що в деяких умовах інтелектуальні фільтри дозволяють ефективно протистояти DDoS атакам, експерти відзначають їх істотний недолік - відсутність чіткої політики блокування пакетів. Зокрема, створення бази легітимних IP адрес з прив'язкою до джерела та блокування всіх нелегітимних запитів на з'єднання порушує працездатність деяких існуючих протоколів, подібних до Mobile IP, тобто при певних параметрах навколишнього середовища система захисту від DDoS атак сама стає причиною DDoS. Таким чином виникає питання про економічну доцільність впровадження того чи іншого механізму захисту та його адекватної експлуатації.

Низькоінтенсивна DDoS-атака характеризується тривалими періодами між передачею пакетів однієї сесії і великим значенням фрагментації пакетів у цій сесії передачі контентної інформації. Можливість проведення низькоінтенсивної атаки пояснюється вразливістю протоколу HTTP та необхідністю обов'язкового очікування сервером кінця передачі POST-запиту. При реалізації низькоінтенсивної DDoS-атаки зловмисник фрагментує POST-запит на пакети малої довжини та відправляє їх серверу з періодичністю менше, ніж значення часу очікування закінчення з'єднання. У результаті сервер змушений очікувати закінчення прийому POST-запитів зловмисника, тоді як запити легітимних користувачів ігноруються через відсутність вільного ресурсу.

Існує велика кількість підходів, що дозволяють провести такі атаки. Кожен їх заснований на використанні специфічних особливої або недоліків однієї з рівнів еталонної моделі взаємодії відкритих систем (open systems interconnection basic reference model, OSI). Кожен рівень містить одну або кілька методик, що дозволяють зробити DoS-атаки на ньому (табл. 1.1).

Таблиця 1.1 – Варіанти DoS атак відповідно до рівня OSI

Рівень моделі	Тип атаки
Прикладний	Атаки на веб-сайти, розсилання спама електронною поштою
Представницький	Спеціально сформовані SSL-запити
Сеансовий	Атаки на telnet
Транспортний	Атаки SYN-пакетами, атаки ICMP-запитами з відомими адресами
Мережений	Атаки ICMP-запитами
Канальний	Атаки пакетами з різними MAC-адресами
Фізичний	Атаки спеціально сформованими пакетами

Атаки рівня інфраструктури

До атак рівня інфраструктури зазвичай відносять атаки на рівнях 3 і 4. Це найбільш поширений тип DDoS-атак, який включає такі вектори, як SYN-флуд, та інші атаки відображення, такі як UDP-флуд. Подібні атаки зазвичай масові і спрямовані на те, щоб перевантажити пропускну здатність мережі або сервери програм. Проте такий тип атак має певні ознаки, тому їх легше виявити.

Атаки рівня програми

До атак рівня додатків зазвичай відносять атаки рівнях 6 і 7. Ці атаки менш поширені, але водночас є складнішими. Як правило, вони не такі масові, як атаки рівня інфраструктури, але націлені на певні дорогі частини програми і призводять до того, що вона стає недоступною для реальних

користувачів. Як приклад, можна привести потік HTTP-запитів на сторінку входу в систему, дорогий API пошуку або навіть потоки XML-RPC Wordpress (також відомі як атаки Wordpress Pingback).

1.2 Аналіз методів виявлення мережевих атак

Методи захисту від DDoS-атак можна умовно класифікувати за двома ознаками. Перша ознака – розташування механізму захисту у мережі. Методи захисту можуть поділятися на джерела, на стороні жертви, а також на проміжних вузлах мережі. Методи, що поєднують різні схеми захисту та забезпечують їхню взаємодію, зазвичай називають гібридними [6]. Прийнято вважати, що гібридні методи забезпечують кращий захист від атак, ніж окремі методи захисту, що працюють самостійно різних ділянках мережі.

Друга ознака – час застосування методу.

Механізми, що застосовуються до настання атаки, належать до методів запобігання [7]. Методи, що використовуються під час атаки, належать до групи виявлення атаки та ідентифікації джерела.

Після виявлення атаки використовуються методи реакції на атаку. Найкращим варіантом є запобігання атакі. Воно може бути досягнуто на всіх етапах шляху трафіку, починаючи від джерела атаки і закінчуючи обробкою даних на стороні сервера, що атакується. Найчастіше використовуються комбіновані засоби запобігання (IPS) та виявлення атак (IDS) – IPDS

Методи, що ґрунтуються на механізмах фільтрації. Існує безліч методів запобігання DDoS-атакам. Для цього дуже часто дослідники пропонують різні механізми фільтрації, наприклад Ingress/Egress filtering [8] , SAVE [9], Hop-Count filtering [10], Route-based filtering [11] та ін. Фільтрування є дуже ефективним способом виявлення підміни IP -Адреса, що особливо актуально в тих випадках, коли використовується посилення або відображення атаки. Різні методи фільтрації, що застосовуються на різних етапах просування трафіку, є потужним інструментом для виявлення фактів заміни адреси.

Найбільшого поширення набув метод Ingress/Egress filtering, оскільки він дозволяє виявити заміну IP-адреси і заблокувати шкідливий пакет ще до того, як він залишить локальну мережу. На основі методів фільтрації трафіку були створені складніші гібридні механізми захисту, такі, як TRACK [12], Active Internet Traffic Filtering (AITF) [13], StopIt [14] та ін. Ці методи виявляють шкідливий трафік і надсилають на маршрутизатори запити на фільтрацію пакетів від підозрілого джерела Кожен із цих методів застосовує різні схеми виявлення шкідливого трафіку та різні методи фільтрації. Однак не всі методи орієнтовані на випадки заміни адреси. Наприклад, метод StopIt застосовує фільтрацію на найближчому до джерела трафіку маршрутизаторі. При цьому реальний шлях трафіку не відстежується, і запит на блокування надходить до джерела, адреса якого вказана у підозрілому пакеті як адреса відправника. У тому випадку, якщо використовувалася заміна адреси, буде здійснюватись фільтрація від джерела, яке насправді може бути легітимним користувачем. Таким чином, слід приділяти увагу проблемі заміни адреси та тим способам фільтрації та відстеження реального шляху трафіку, які дозволяють її виявити. Проте ці методи можна успішно використовувати у складі комплексних рішень щодо протидії атакам.

Математичні методи та методи інтелектуального аналізу даних. Крім описаних методів для аналізу трафіку можуть бути реалізовані механізми захисту на основі математичних методів, наприклад визначення ентропії [15].

В останні роки активно розробляються методи захисту, що базуються на різних алгоритмах інтелектуального аналізу даних. Як приклади можна навести механізми з урахуванням методу найближчих сусідів (kNN) [16], які навчаються нейронних мереж [17]. Такі методи можуть бути застосовні і для протидії атакам, що використовують відображення трафіку та його посилення. Інтелектуальні методи аналізу даних дозволяють виявити різні девіації трафіку, і навіть підозріле поведінка клієнтів. Складність полягає в тому, що навчання нейронних мереж, наприклад, може зайняти тривалий час.

У роботі [18] було запропоновано статистична модель виявлення DDoS-атак, здійснюваних за протоколом TCP. Модель аналізує прапори в заголовку кожного пакета та порівнює реальний трафік із заданим шаблоном нормального трафіку. Відхилення від шаблонного трафіку оцінюються як аномалія.

Метод, представлений у роботі [19], заснований на принципі асиметрії трафіку у разі атаки. Як шаблон нормального трафіку прийнята схема симетричного обміну запитами клієнта та відповідями сервера. У разі значного підвищення кількості запитів або відповідей на запити, які не можуть бути коректно оброблені, порушується симетрія трафіку, і така ситуація розцінюється як атака. Варто відзначити, що аналіз симетричності трафіку пропонується також як метод боротьби з атаками, заснованими на відображенні трафіку і його посиленні, так як при даних атаках асиметрія трафіку є значною і таку атаку стає легко виявити.

Метод захисту однорангових мереж від різних атак, вкладених у інфраструктуру комп'ютерної мережі, представлений у роботі [20].

Пропонований механізм захисту встановлюється на прикордонному маршрутизаторі мережі, яку спрямовано атака. Метод виявлення атаки заснований на статистичному аналізі трафіку та його зіставленні із шаблонним трафіком. Після виявлення атаки використовується схема маркування пакетів з метою виявлення джерела атаки та блокування трафіку від нього. Така схема є одним з ефективних методів захисту і у випадку, коли використовується підміна адреси, так як відстежується реальний шлях трафіку, а не блокування трафіку від зазначеного в пакеті відправника.

Механізми захисту від атак, які використовують відображення трафіку. Розглянемо механізми захисту, розроблені безпосередньо для атак, заснованих на відображенні та посиленні трафіку.

Загальний механізм запобігання відображенню трафіку RAD (Reflector Attack Defense) пропонується у [21]. В основі методу використовується message authentication code (MAC). Коли той або інший вузол надсилає запит,

він поміщає MAC у поле. У відповіді на це повідомлення розміщується той же MAC, що і в запиті. Вузол, отримавши відповідь на запит, звіряє MAC з надісланого ним запиту та MAC з отриманої відповіді. Якщо MAC збігаються, повідомлення надсилається. Інакше вважається, що відповідь є відбитий трафік і повідомлення відхиляється.

В даний час розроблено безліч методів захисту від DNS-атак. У роботі [22] в основі методу захисту DAAD (DNS Amplification Attacks Detector) лежить той факт, що при атаці DNS вузол, що атакується, отримує велику кількість відповідей на відправлені раніше запити. Автори пропонують вести базу адрес DNS-серверів, на які надсилалися запити від того чи іншого вузла. Усі відповіді повинні перевірятися, і якщо вхідний пакет дійсно є відповіддю на запит, він буде прийнятий. Якщо з вузла, якому адресована відповідь, не надсилався DNS-запит на даний сервер, такий пакет має бути відхилений.

Цей метод дуже ефективний, проте слід пам'ятати, що ведення подібних баз потребує великого обсягу ресурсів.

У роботі [23] пропонується встановлення попереднього DNS-резольвера та створення тунелю, що використовує протоколи IPSec або SSL, між попереднім резольвером та DNS-резольвером на стороні організації. Всі запити DNS проходять виключно через тунельований канал зв'язку і не можуть надходити безпосередньо із зовнішніх джерел. Зазначається, що основна фільтрація DNS-відповідей має здійснюватись провайдером послуг інтернет-зв'язку після відповідного запиту від організації. Тунелювання у разі грає лише допоміжну роль.

У роботі [24] пропонується механізм RRL (Response Rate Limiting), спрямований обмеження кількості унікальних відповідей від DNS-сервера. Цей механізм захисту використовується на стороні DNS-сервера і аналізує вихідний трафік, повністю ігноруючи вхідний. Суть методу у тому, що адреси, куди було відправлено відповідь, записуються.

При цьому визначається обмеження кількості відповідей сервера на кожну адресу. Якщо це число перевищено, відповіді на цю адресу більше не

надсилаються. Такий метод ефективний зниження потоку шкідливого трафіку від сервера, але при цьому існує ймовірність помилки першого роду. Однак слід пам'ятати, що не завжди власники DNS-серверів готові використовувати такі механізми. Ведення бази адрес вимагає певних ресурсів. Крім того, не завжди власники таких серверів стурбовані тим, що їх сервери використовуються для атак, а також не завжди помічають збільшення навантаження на сервер у періоди їх використання для реалізації атак.

У роботі [25] пропонується метод FB (Flow-based), що ґрунтується на виявленні девіації трафіку щодо шаблонного. Аналізується кількість вхідних пакетів за протоколом DNS та їх розмір. Якщо кількість та розмір пакетів перевищують задані значення, така ситуація розцінюється як атака. При цьому автор зазначає, що в ситуаціях, коли здійснюється атака, але при цьому параметри, що аналізуються, не перевищують порогових значень, трафік вважається легітимним. Отже, відсоток помилок другого роду окремих випадках може бути дуже високий. Цей метод, проте, дуже перспективний при відповідному доопрацюванні.

У таблиці 1.2 наведено загальний порівняльний аналіз розглянутих способів захисту.

Існуючі методи виявлення DDoS-атак дозволяють ефективно розпізнавати DDoS-атаки транспортного і мережевого рівнів і малоефективні для виявлення низькоінтенсивних DDoS-атак прикладного рівня, так як не враховують особливостей низькоінтенсивної атаки. Виняток становлять методи на основі м'яких рішень, але і вони мають низку істотних недоліків, що знижують ефективність їх застосування, головний з яких – необхідність попереднього навчання. З іншого боку, точність функціонування методів з урахуванням м'яких рішень залежить від якості наборів характеристик атаки. Чим більше навчання аналізатора використовується значень з простору характеристик атаки, тим точніше її визначення (менше помилок 1 і 2 роду). Однак навчання по всьому простору значень характеристик атаки неможливе

через обчислювальну та тимчасову складність реалізації такого навчання. На додаток до цього навчання аналізатора атаки стає безперспективним у разі випадкового характеру динаміки характеристик низькоінтенсивної атаки, оскільки трафік атаки малої потужності стає неперіодичним, що нетиповою її ознакою [3-5].

Таблиця 1.2 – Порівняльний аналіз способів захисту

Метод	Місце впровадження	Переваги	Недоліки
RAD	На стороні вузла, що атакується	Висока точність виявлення зловмисного трафіка	Необхідність зберігання запитів та відповідних їи MAC до отримання відповіді. Необхідність розміщення MAC у відповіді на запит
DAAD	На стороні вузла, що атакується	Достатньо висока точність виявлення зловмисного трафіка	Необхідність ведення бази даних, вживання великого об’єму ресурсів
Тунелювання	На стороні вузла, що атакується	Безпечна передача даних	Грає додаткову роль оскільки основна фільтрація проводиться інтернет-провайдером
RRL	На стороні сурвера	Зниження потоку зловмисного трафіка	Необхідність ведення баз даних на сервері. Ймовірність помилки першого роду
FB	На стороні вузла, що атакується	Достатньо висока точність виявлення зловмисного трафіка	У окремих випадках відсоток помилок другого роду достатньо великий

1.3 Файли трафіку та їх використання для виявлення мережевих атак

Для розробки технології виявлення мережевих атак необхідно провести аналіз трафіку під час атак різного типу та звичайного (чистого) трафіку. Мережеве обладнання усіх відомих фірм розробників має власні протоколи, що дозволяються зберігати трафік для аналізу.

NetFlow – це протокол, розроблений компанією Cisco Systems, також відомий як стандарт IPFIX – компонент Cisco IOS. Протокол NetFlow працює на IP-пристроях (маршрутизаторах, комутаторах 3-го рівня) та надає статистику IP-трафіку. Ряд виробників упровадили цей стандарт (напр., Juniper випустила аналогічне рішення під назвою jFlow). Інші компанії (напр., HP, Foundry та Extreme) використовують технологію контролю потоку даних sFlow.

Незважаючи на назву, NetFlow охоплює значний обсяг даних і є багатим джерелом інформації, що оновлюється в режимі реального часу; ця інформація доступна в будь-який час і містить багато відомостей про трафік даних мережі.

Система візуалізує не тільки параметри TCP/IP на 3-му та 4-му рівні (IP-адреса джерела, цільова IP-адреса, протокол, порт), але й додаткові атрибути трафіку (Type of Service, DSCP, ідентифікатор джерела та цільові області) автономних систем у протоколі BGP) та додаткову інформацію про маршрутизацію та трафік - наступна транзитна ділянка, вхідна та вихідні інтерфейси, мережна адреса джерела та цільова мережна адреса.

Технологія NetFlow дозволяє виявляти проблеми, вузькі місця в мережі, перевіряти налаштування класів трафіку (CoS/ToS), визначати відправлений трафік та програми. За допомогою NetFlow можна створити відносно недорогу та просту систему моніторингу мережевого трафіку.

за допомогою NetFlow можна моніторити будь-який канал мережі. Оскільки налаштування NetFlow на маршрутизаторі виконується на

програмному рівні, можна вибірково активувати його моніторингові пристрої, а також інші критичні пристрої (напр., на концентраторі, на маршрутизаторі, який забезпечує підключення до Інтернету або в місцях, де виникають проблеми в мережі).

Більше того, NetFlow - це відкритий протокол, тому він підтримує ряд сторонніх програм для моніторингу мережі в режимі реального часу, створення звітів та зіставлення користувачів у мережі. Зазвичай програми розробляються під індивідуального клієнта та конкретні вимоги.

Структура пакетів NetFlow наведена на рис.1.1.

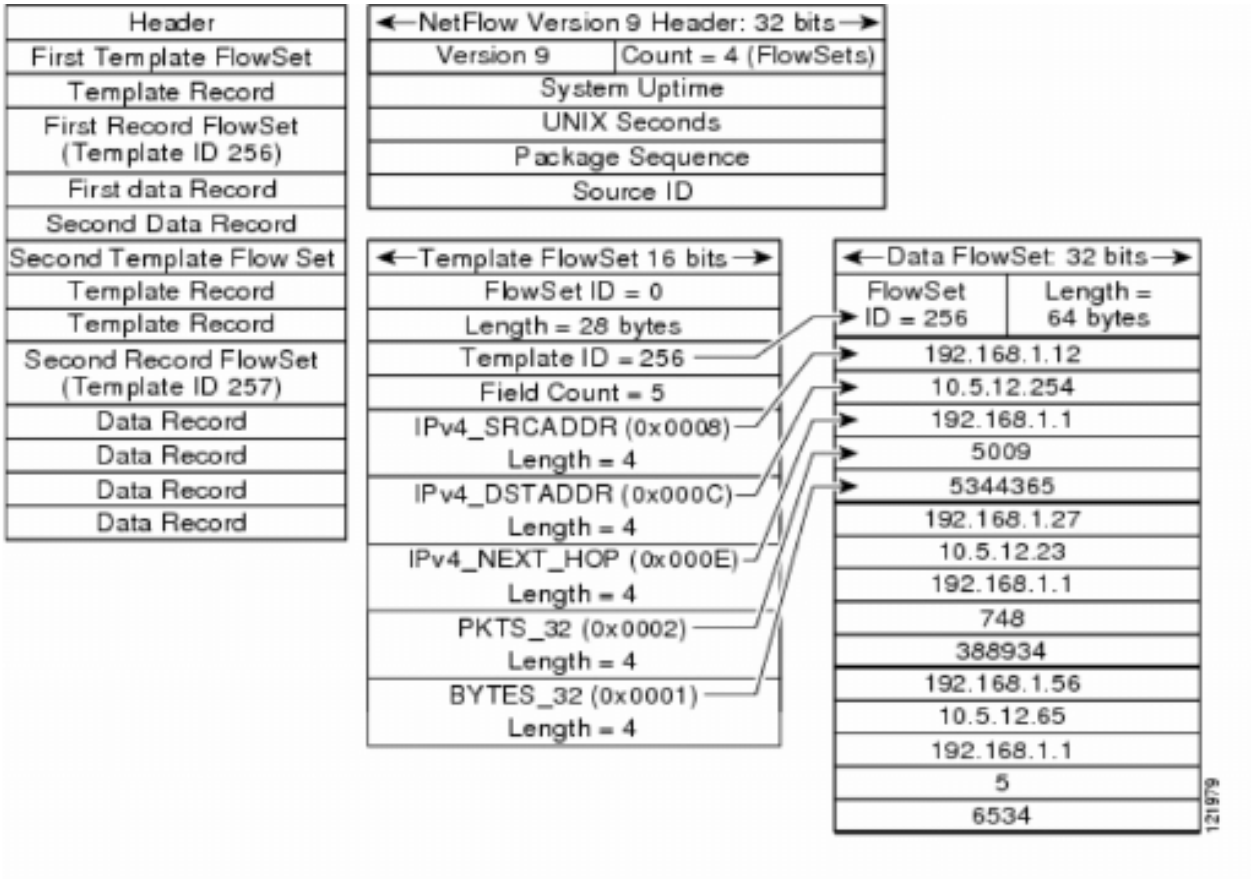


Рисунок 1.1 – Структура пакетів NetFlow

Отже аналіз файлів протоколу NetFlow надає можливість провести класифікацію низкоінтенсивних мережевих атак та виявити основні ознаки атаки при використанні штучного інтелекту (нейронних мереж), що дозволить запропонувати технологію виявлення атак в режимі реального часу з подальшим розриванням з’єднання з атакуючою точною.

Однак, без відповідних інструментів для обробки даних NetFlow не особливо корисний. Отже в роботі додатково пропонується розробка програмного забезпечення що дозволить обробляти NetFlow пакетів під атаками та звичайних пакетів з метою виявлення особливостей різних типів атак.

2. МАТЕМАТИЧНІ ТА АЛГОРИТМІЧНІ ОСНОВИ ВИЯВЛЕННЯ МЕРЕЖЕВИХ АТАК

2.1 Підходи до визначення стану атаки на мережу

Перед розробкою технології виявлення мережевих атак необхідно обрати підхід для використання під час реалізації.

У табл.2.1 наведено порівняння підходів до виявлення мережевих атак.

Таблиця 2.1 – Порівняння підходів виявлення

Підхід (Спосіб)	Переваги	Недоліки
Статистичний	Адаптація к поводженню суб’єкту. Не потрібні знання про можливі атаки та вразливостях, що використовуються	1. Можливість маніпуляції еталонними профілями параметрів трафіка 2. Нечутливість до порядку слідування подій 3. Обєктивна важкість визначення межевих значень характеристик, що відстежуються
Використання експертних систем	Відсутність хибних тривог	Необхідність постійного оновлення сигнатур (актуалізація)
Нейромережеві аналізатори та генетичні алгоритми	Здатність вивчати характеристики атак та ідентифікувати елементи, які на схожі на те, що спостерігалось раніше	Точність виявлення атак залежиться від якості навчання нейромережі

Продовження

Підхід (Спосіб)	Переваги	Недоліки
Використання Data Mining та машинного навчання	Малий час виявлення події атаки	1. Недостатньо протестований на практиці 2. Потребує достатньо великих обчислювальних потужностей

В даний час найбільш ефективним способом реалізації виявлення низькоінтенсивної атаки за пропонованим показником є аналізатор на основі гібридної нейронної мережі, що складається з фільтрів заборони та нейронної мережі зі зворотним розповсюдженням на основі зірки Гроссберга (рис. 2.1)

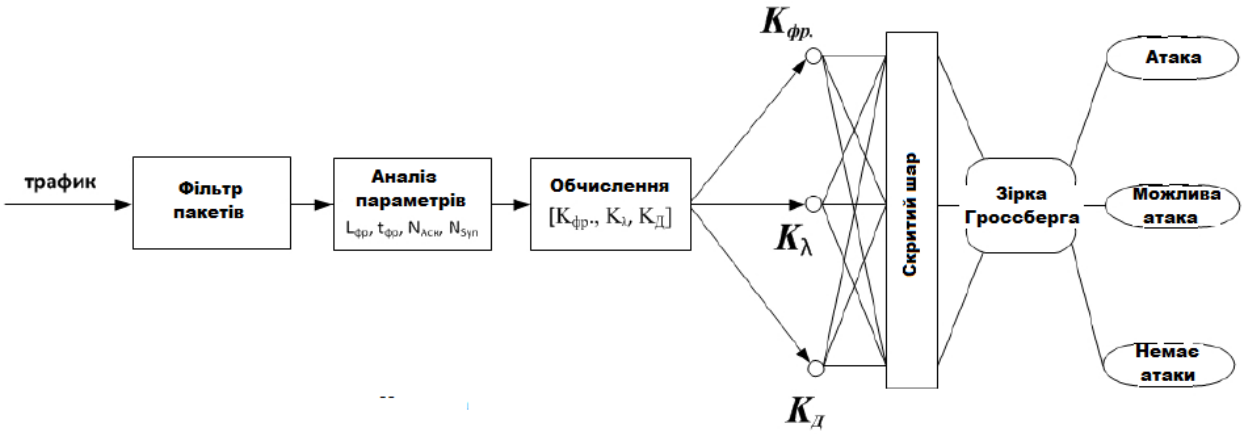


Рисунок 2.1 – Гібридна нейронна мережа зі зворотним розповсюдженням на основі зірки Гроссберга

Така реалізація дозволяє звузити безліч можливих значень показника тривоги на вході нейронної мережі до трьох чітких станів: «Атака», «Можливість атаки», «Відсутність атаки» (рис. 4), однак має такі недоліки:

- необхідність попереднього знання характеристик атак попереднього навчання нейросети;

- проблема в точному визначенні межі областей станів наявності та відсутності атаки, що в сукупності визначає значення ризику не виявлення атаки.

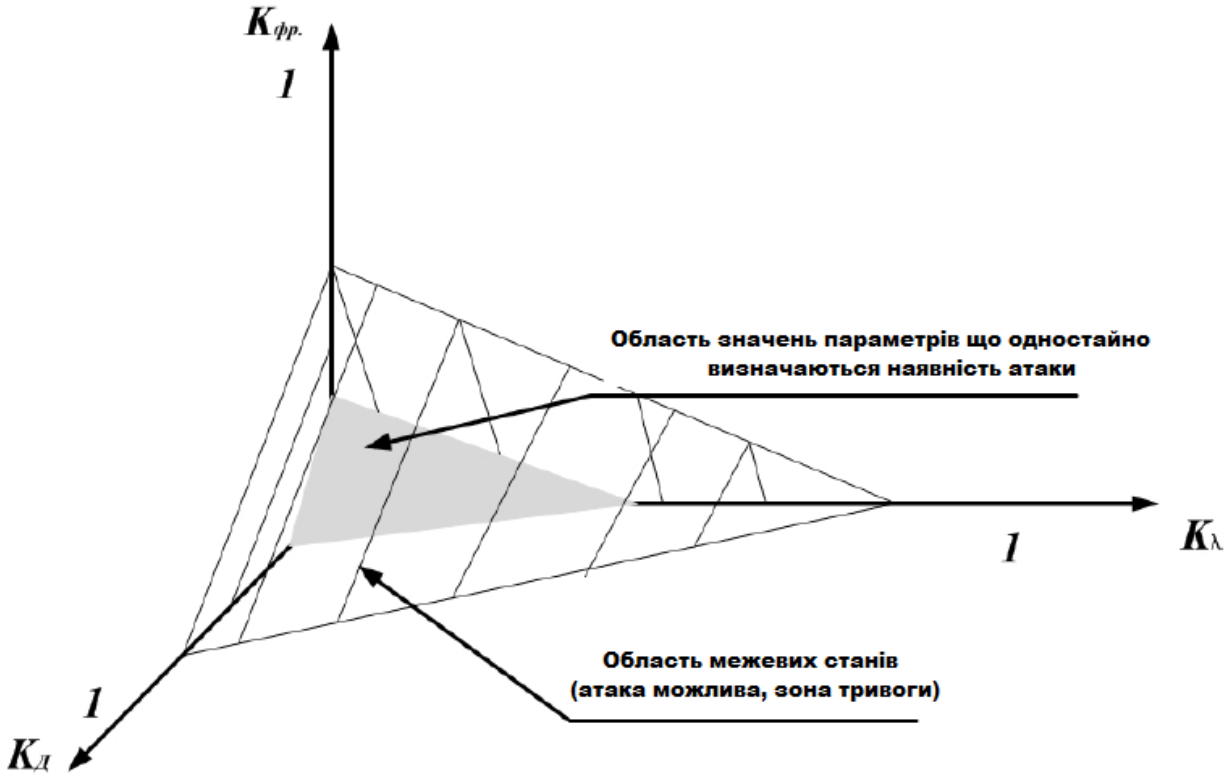


Рисунок 2.2 – Області можливих станів аналізатора атаки

Для усунення зазначених недоліків пропонується додатково аналізувати швидкість динаміки градієнтного руху показника тривоги (рис. 2.3).

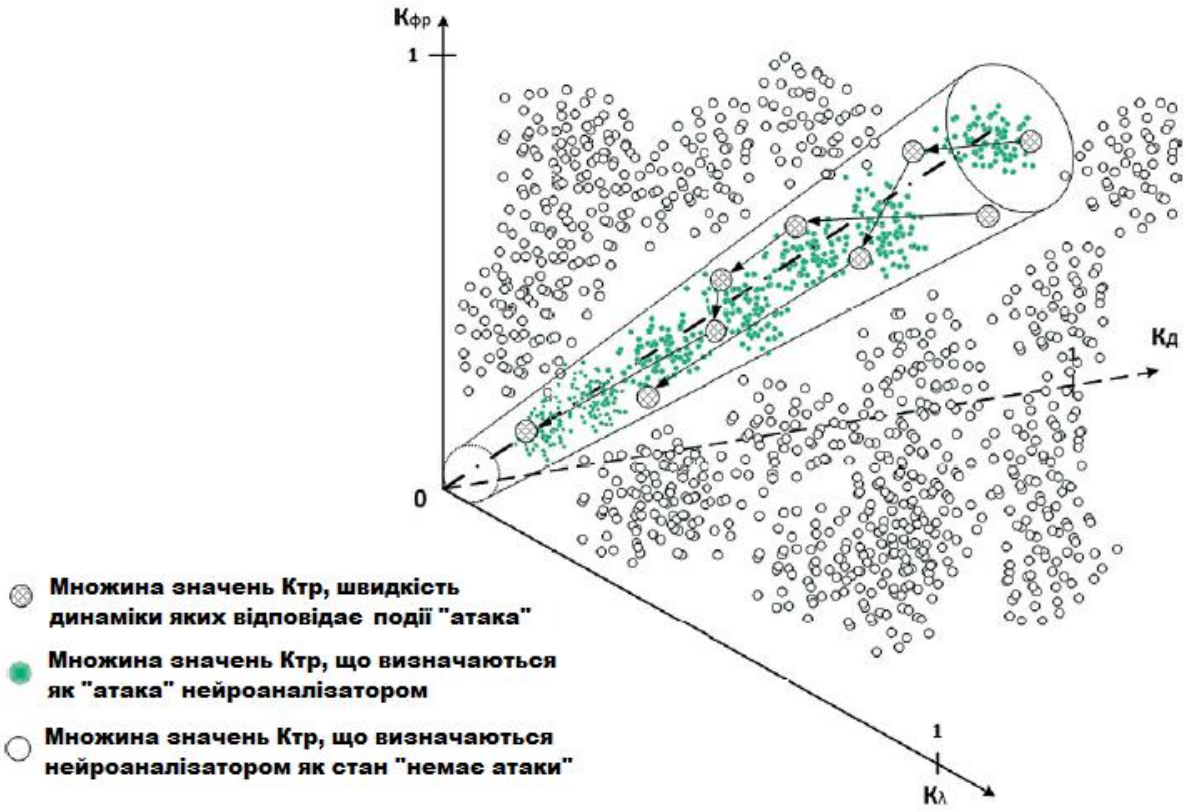


Рисунок 2.3 – Динаміка градієнтного руху показника тривоги

2.2 Підходи засновані на машинному навчанні

Машинне навчання (ML) викликало широкий інтерес до багатьох додатків і областей дослідження, зокрема в кібербезпеці. Оскільки обладнання та обчислювальна потужність стають доступнішими, методи машинного навчання можна використовувати для аналізу та класифікації поганих акторів із величезного набору доступних даних.

Підходи до навчання з наглядом застосовуються в контексті класифікації, де вхідні дані збігаються з виходом, або регресії, коли вхід зіставляється з безперервним результатом. Неконтрольоване навчання здебільшого здійснюється за допомогою кластеризації та застосовується для дослідницького аналізу та зменшення розмірів. Обидва ці підходи можна застосувати в кібербезпеці для аналізу зловмисного програмного забезпечення майже в реальному часі, таким чином усуваючи слабкі сторони традиційних методів виявлення.

Застосування машинного навчання для виявлення бот-мереж широко досліджувалося. Стеванович та Педерсен в 2014 розробили систему виявлення бот-мереж на основі потоку з використанням контрольованого машинного навчання. Сантана, Сутхахаран і Моханті в 2018 дослідили кілька моделей машинного навчання, щоб охарактеризувати їхні можливості, продуктивність та обмеження для атак бот-мереж.

Машинне навчання також розглядалося як рішення для оцінки NetFlows або даних, пов’язаних із IP, де основним питанням був би вибір параметрів, які могли б досягти високої якості результатів

У кількох роботах про виявлення на основі NetFlow використовується ряд методів машинного навчання. Kozik, Pawlicki та Michal в 2018 представили розподілений ELM, Random Forest і Gradient-Boosted Trees як чутливі до витрат підходи до кібербезпеки.

Extreme Learning Machine (ELM) — це алгоритм навчання, який використовує нейронні мережі з прямим зв’язком з одним або кількома шарами прихованих вузлів. Ці приховані вузли налаштовуються випадковим чином, і їх відповідні вихідні ваги аналітично визначаються алгоритмом. За словами розробників, цей алгоритм навчання може забезпечити хорошу продуктивність узагальнення і може навчатися в тисячу разів швидше, ніж звичайні алгоритми навчання для нейронних мереж із прямим зв’язком.

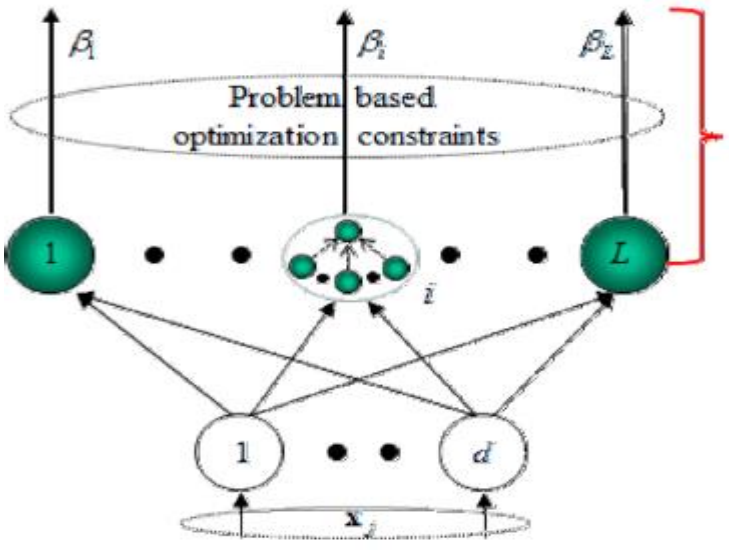


Рисунок 2.4 – Спрощена ілюстрація алгоритму ELM

Random Forest (RF) — це контрольований алгоритм машинного навчання, який передбачає використання кількох дерев рішень для виконання завдань класифікації та регресії. Алгоритм Random Forest вважається ансамблевим алгоритмом машинного навчання, оскільки він включає концепцію голосування більшістю кількох дерев. Вихід алгоритму, представлений у вигляді передбачення класу, визначається із сукупного результату всіх класів, передбачених окремими деревами. Нещодавні дослідження досліджували можливості Random Forest в атаках на безпеку, зокрема в ін'єкційних атаках, фільтрації спаму, виявленні шкідливих програм тощо.

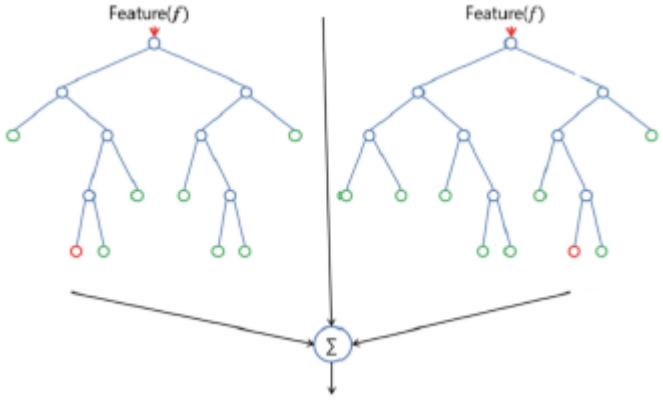


Рисунок 2.5 – Спрощена ілюстрація алгоритму випадкового лісу

Підвищення градієнта — це техніка машинного навчання для задач регресії та класифікації, яка створює модель прогнозування у вигляді ансамблю слабких моделей прогнозування, як правило, дерев рішень. Він поєднує в собі елементи функції втрати, слабкого учня та адитивну модель. Модель додає слабких учнів, щоб мінімізувати функцію втрат. Основне припущення з підвищенням градієнта полягає в тому, щоб багаторазово використовувати закономірності в залишках і посилити модель зі слабкими прогнозами та покращити її. Коли він досягає стадії, коли залишки не мають жодного шаблону, який можна було б змоделювати, моделювання залишків буде припинено (інакше це може призвести до переобладнання).

Математично це означає мінімізацію функції втрат таким чином, що втрата тесту досягає свого мінімуму.

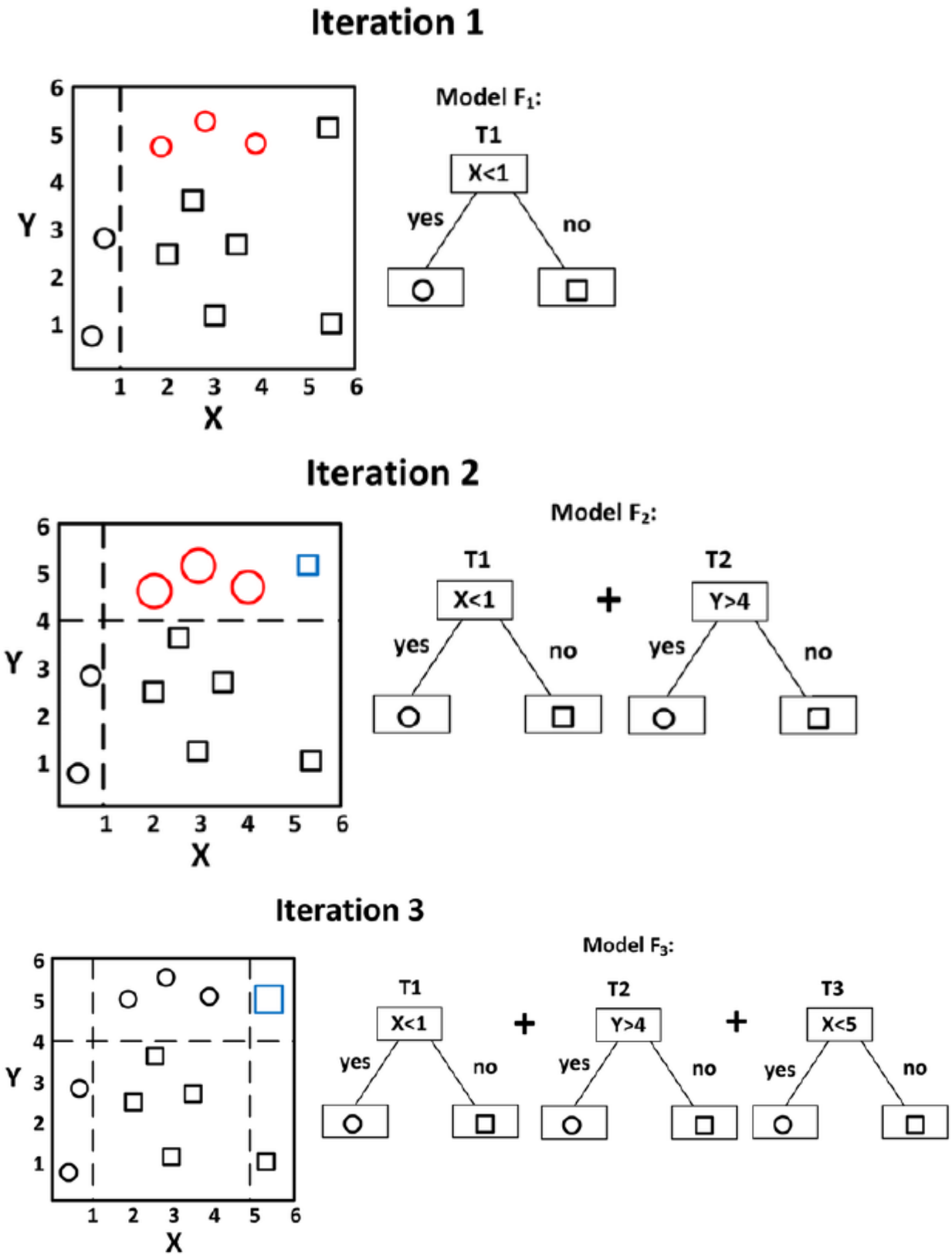


Рисунок 2.6 – Спрощена ілюстрація алгоритму посилення градієнта

Машина опорних векторів (SVM) — це контрольована модель навчання, яка використовується для аналізу регресії та класифікації. Він є дуже бажаним через його високу точність при меншій обчислювальній потужності та складності. SVM також використовується в комп'ютерній безпеці, де вони використовуються для виявлення вторгнень. Наприклад, один клас SVM використовувався для аналізу записів на основі нової функції ядра і точної класифікації інтернет-трафіку.

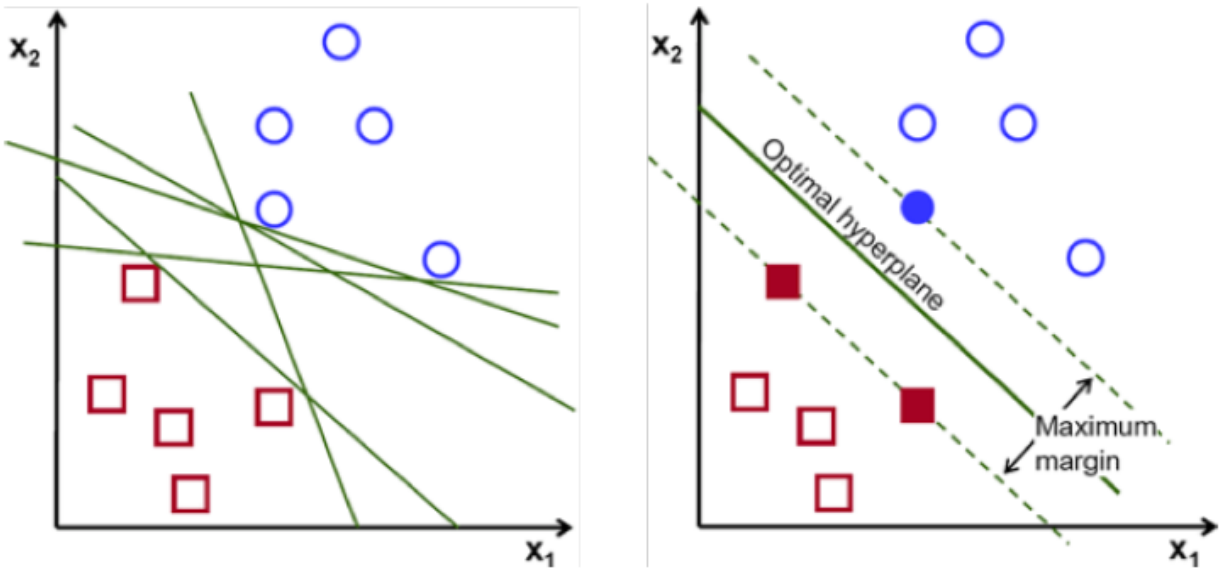


Рисунок 2.7 – Спрощена ілюстрація машини опорних векторів

Логістична регресія — це контрольована модель навчання, яка використовується як метод бінарної класифікації. Сам термін запозичений зі статистики. В основі методу використовуються логістичні функції, сигмовидна крива, яка корисна для низки областей, включаючи нейронні мережі. Логістична регресія моделює ймовірність проблем класифікації з двома можливими результатами. Логістичну регресію можна використовувати, щоб визначити мережевий трафік як шкідливий чи ні (Bapat et al. 2018).

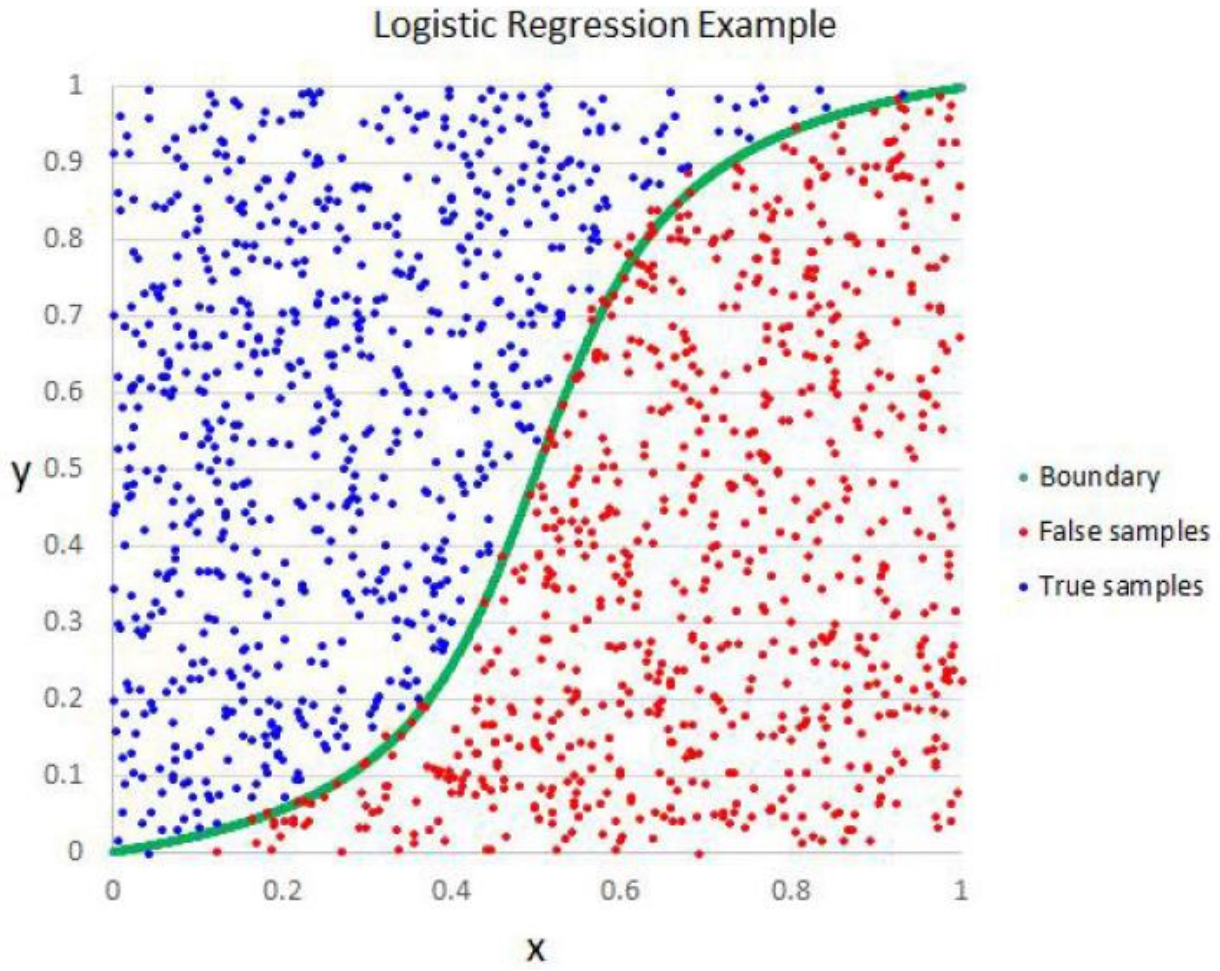


Рисунок 2.8 – Спрощена ілюстрація машини опорних векторів

2.3 Структура NetFlow

Традиційні методи виявлення аномалій, такі як виявлення вторгнень і глибока перевірка пакетів (DPI), зазвичай вимагають необроблених даних або підписів, опублікованих виробниками. DPI забезпечує більш точні дані, але це також дорожче обчислень. DPI не працює із зашифрованими даними. Він також є предметом проблем щодо конфіденційності, оскільки містить конфіденційну інформацію користувачів. З огляду на тенденції, спрямовані на конфіденційність даних і шифрування, необроблені дані також можуть бути нелегко доступними

З іншого боку, дані Netflow не містять такої конфіденційної інформації і широко використовуються операторами мереж. При належних методах

аналізу дані NetFlow можуть бути багатим джерелом інформації для виявлення аномалій.

З тих пір NetFlow став галузевим стандартом для збору даних сеансу. Дані NetFlow надають інформацію, яку можна використовувати для визначення використання мережевого трафіку та стану ресурсів, а також виявлення аномалій мережі та потенційних атак.

Потоки визначаються як односпрямована послідовність пакетів з деякими загальними властивостями, що проходять через мережевий пристрій. Записи потоків включають таку інформацію, як IP-адреси, кількість пакетів і байтів, мітка часу, тип послуги (ToS), порти додатків, інтерфейси введення та виведення тощо. Кортеж із 5, що складається з IP-адреси клієнта, номера порту клієнта, IP-адреси сервера, номера порту сервера та протоколу, включених у дані потоку, важливий для ідентифікації з'єднання. Досліджуючи кореляцію між потоками, ми можемо знайти значущі моделі трафіку та поведінку вузлів.

Вихідні дані NetFlow мають такі атрибути:

- StartTime – час початку записаного потоку;
- Dur – тривалість потоку;
- Proto – використовуваний протокол (TCP, UDP тощо);
- SrcAddr – IP-адреса джерела;
- Sport – Source Port;
- Dir – напрям комунікації;
- DstAddr – адреса призначення;
- Dport – порт призначення;
- State – стан протоколу;
- sTos – джерело Тип послуги;
- dTos – тип послуги призначення;
- TotPkts – загальна кількість обмінених пакетів;
- TotBytes – загальна кількість обмінених байтів;
- SrcBytes – кількість байтів, надісланих джерелом;

- Label – мітка, призначена цьому мережевому потоку.

Для проведення експериментів нам необхідна виборка даних бажано помічена стосовно того які атаки відбувались.

Для файлів NetFlow існують набори трафіка у вільному доступі

Набір даних CTU-13 — це маркований набір даних, який використовується в літературі для навчання алгоритмів виявлення бот-мереж. Він був створений Університетом CTU 2011 і фіксує реальний трафік ботнету, змішаний зі звичайним трафіком і фоновим трафіком.

Що стосується числових характеристик NetFlow (у нашому випадку: тривалість зв’язку, загальна кількість обмінених байтів, кількість байтів, надісланих джерелом), витягується 5 функцій:

- сума;
- середнє;
- стандартне відхилення;
- максимум;
- медіана.

Усі ці вилучені функції дозволять нам навчати різні моделі. Однак вони можуть бути зайвими або не має значення для виявлення ботнетів, тому потрібен вибір функції.

3. НАПИСАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір мови програмування

Мовою для написання програмного забезпечення, був обраний Python, завдяки перевагам і простоті стосовно будь-якого іншого. Вибір цієї мови обумовлений тим, що він є в першу чергу, розрахованим для математичних розрахунків.

Python - високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника та читання коду. Синтаксис ядра Python мінімальний. У той же час, стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, у тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне та аспектно-орієнтоване. Основні архітектурні риси - динамічна типізація, автоматичне управління пам'яттю, повна інтроспекція, механізм обробки винятків, підтримка багатопоточних обчислень та зручні високорівневі структури даних. Код Python організовується в функції і класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, який підтримує більшість платформ, що активно використовуються. Він розповсюджується під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких програмах, включаючи пропрієтарні. Існують реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM та інших. Проект PyPy пропонує реалізацію Python за допомогою JIT-компіляції, яка значно збільшує швидкість виконання Python-програм.

Python — мова програмування, що активно розвивається, нові версії (з додаванням/зміною мовних властивостей) виходять приблизно раз на два з

половиною роки. Внаслідок цього та деяких інших причин на Python відсутні стандарти ANSI, ISO або інші офіційні стандарти, їх роль виконує CPython.

Python портований і працює майже на всіх відомих платформах від КПК до мейнфреймів. Існують порти під Microsoft Windows, практично всі варіанти UNIX (включаючи FreeBSD та Linux), Plan 9, Mac OS та Mac OS X, iPhone OS 2.0 та вище, Palm OS, OS/2, Amiga, HaikuOS, AS/400 і навіть OS /390, Windows Mobile, Symbian та Android.

При цьому, на відміну від багатьох систем, що портуються, для всіх основних платформ Python має підтримку характерних для даної платформи технологій (наприклад, Microsoft COM/DCOM). Більше того, існує спеціальна версія Python для віртуальної машини Java - Jython, що дозволяє інтерпретатору виконуватися на будь-якій системі, яка підтримує Java, при цьому класи Java можуть безпосередньо використовуватися з Python і навіть бути написаними на Python. Також кілька проектів забезпечують інтеграцію з платформою Microsoft .NET, основні з яких IronPython і Python.Net.

Python підтримує динамічну типізацію, тобто тип змінної визначається лише під час виконання. Тому замість «надання значення змінної» краще говорити про «зв'язування значення з деяким ім'ям». У Python є вбудовані типи: булевий, рядок, Unicode-рядок, ціле число довільної точності, число з плаваючою комою, комплексне число та деякі інші. З колекцій Python вбудовані: список, кортеж (незмінний список), словник, безліч та інші. Усі значення є об'єктами, зокрема функції, методи, модулі, класи.

Додати новий тип можна або написавши клас (class), або визначивши новий тип у модулі розширення (наприклад, написаному мовою C). Система класів підтримує успадкування (одинкове та множинне) та метапрограмування. Можливе успадкування від більшості вбудованих типів та типів розширень.

Дизайн мови Python побудований навколо об'єктно-орієнтованої моделі програмування. Реалізація ООП у Python є елегантною, потужною та добре

продуманою, але водночас досить специфічною порівняно з іншими об'єктно-орієнтованими мовами.

Програмне забезпечення (додаток або бібліотека) на Python оформляється у вигляді модулів, які можуть бути зібрані в пакети. Модулі можуть розташовуватися як у каталогах, так і ZIP-архівах. Модулі можуть бути двох типів за своїм походженням: модулі, написані на «чистому» Python, та модулі розширення (extension modules), написані іншими мовами програмування. Наприклад, у стандартній бібліотеці є «чистий» модуль pickle та його аналог на Ci: cPickle. Модуль оформляється як окремого файлу, а пакет — як окремого каталогу. Підключення модуля до програми здійснюється оператором import. Після імпорту модуль представлений окремим об'єктом, що дає доступ до простору назв модуля. Під час виконання програми модуль можна перезавантажити функцією reload().

Багата стандартна бібліотека є однією з найпривабливіших сторін Python. Тут є засоби для роботи з багатьма мережними протоколами та форматами Інтернету, наприклад, модулі для написання HTTP-серверів та клієнтів, для розбору та створення поштових повідомлень, для роботи з XML тощо. Набір модулів для роботи з операційною системою дозволяє писати кросплатформні програми. Існують модулі для роботи з регулярними виразами, текстовими кодуваннями, мультимедійними форматами, криптографічними протоколами, архівами, серіалізації даних, підтримка юніт-тестування та ін.

Крім стандартної бібліотеки існує безліч бібліотек, що надають інтерфейс всім системним викликам на різних платформах; зокрема, на платформі Win32 підтримуються всі виклики Win32 API, а також COM в об'ємі не меншому, ніж у Visual Basic або Delphi. Кількість прикладних бібліотек для Python в різних областях без перебільшення величезна (веб, бази даних, обробка зображень, обробка тексту, чисельні методи, додатки операційної системи і т. д.).

3.2 Написання програмного коду

Основна мета розробки програмного забезпечення — виявити шкідливе програмне забезпечення або трафік бот-мереж із набору даних NetFlow за допомогою різних підходів машинного навчання.

Програмна реалізація запропонованої техніки спрямований на:

- виявлення зловмисного програмного забезпечення або трафіку бот-мережі з даних Netflow (Система повинна приймати будь-який набір даних Netflow будь-якого розміру, чистий або зі шкідливим програмним забезпеченням, і класифікувати як звичайний або атакуючий трафік);
- порівняння різноманітних методів машинного навчання та рекомендацію відповідного для конкретних випадків використання.

Крім базового набору функцій мови Python при написанні програмного забезпечення аналізу мережевих атак були використані наступні додаткові бібліотеки:

- Numpy
- Pandas
- Scipy
- Datetime
- h5py
- Matplotlib
- Scikit-learn
- Tensorflow (у модулі predict_neural_network_stat_analysis.py)

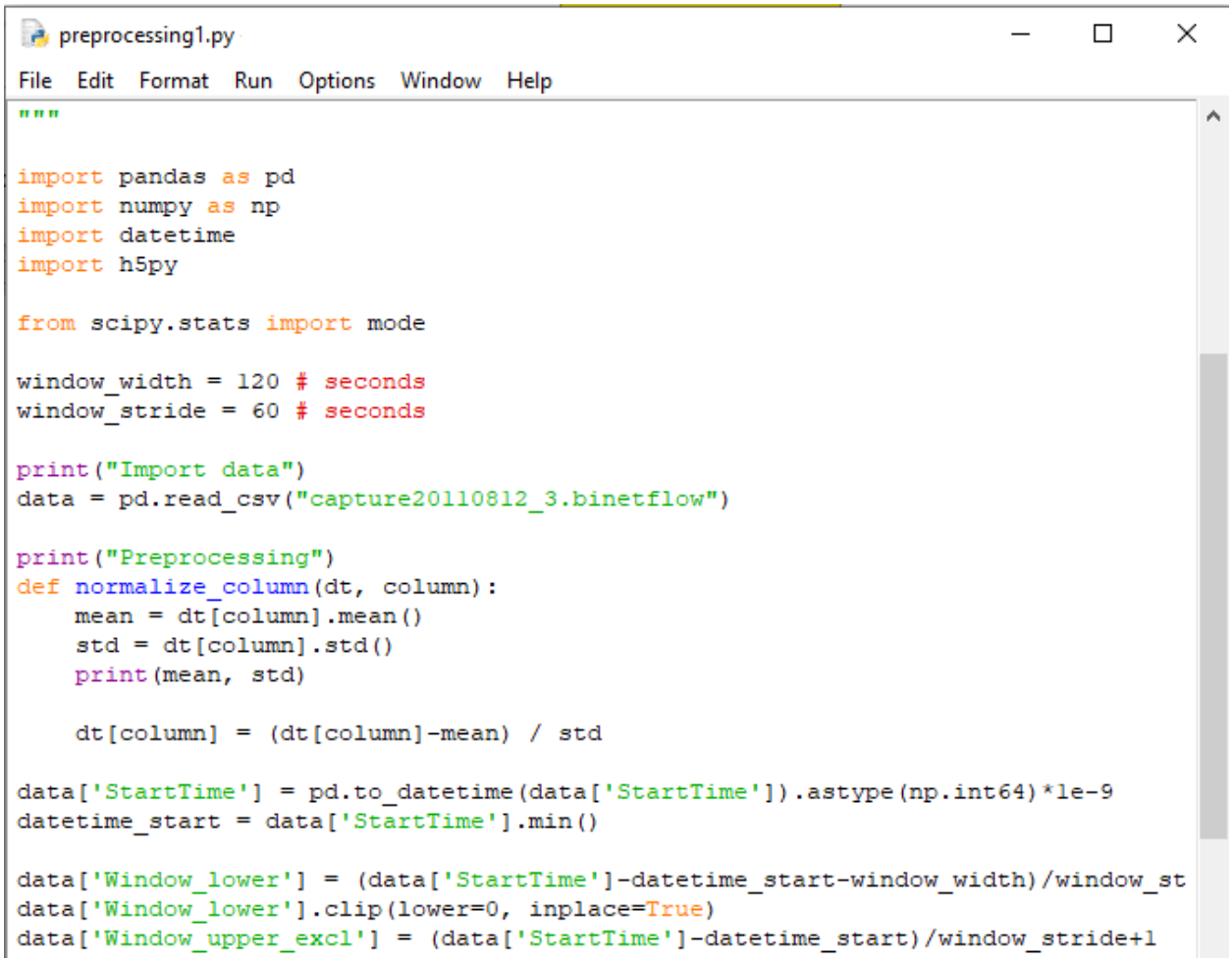
Опис файлів

preprocessing1.py і preprocessing2.py – це файли, які використовуються для вилучення значущих даних із необроблених файлів netflow.

feature_extraction.py і pca_tsne.py намагаються зменшити кількість об'єктів за допомогою вбудованих методів або методів зменшення розмірності.

predict_random_forest_bootstrap.py і predict_svm.py реалізують класифікатор для виявлення шкідливого програмного забезпечення за допомогою Random Forest і Support Vector Machine.

predict_gradient_boosting_stat_analysis.py, predict_logistic_reg_stat_analysis.py, predict_neural_network_stat_analysis.py і predict_statistic_analysis_bootstrap.py здійснюють статистичний аналіз різних класифікаторів: Gradient Boosting, Logistic Regression, Forest Neural Network з bootstrap і boostrap.



4. РЕЗУЛЬТАТИ ОБРОБКИ ДАНИХ

Аналіз трафіка отриманого на основі файлів протоколу NetFlow за допомогою написаного програмного забезпечення дав результати, що наведені у цьому розділі. Ці результати надали можливість порівняти наявні алгоритми.

Логістична регресія

Метод логістичної регресії — це алгоритм, який використовує лінійну комбінацію ознак для класифікації потоків. Як показано в Розділі 4.3.2, для налаштування параметрів моделі необхідна перехресна перевірка. Кінцеві вибрані параметри: C = 550 і $Weight_{non\text{-}botnet}$ = 0,044.

Машина опорних векторів

Метод опорних векторних машин — це алгоритм, який використовує ядра для перетворення простору даних і потім пробує знайти окремий рядок, щоб розділити дані на два класи. Для лінійного ядра вибраний параметр = 10^{-9}.

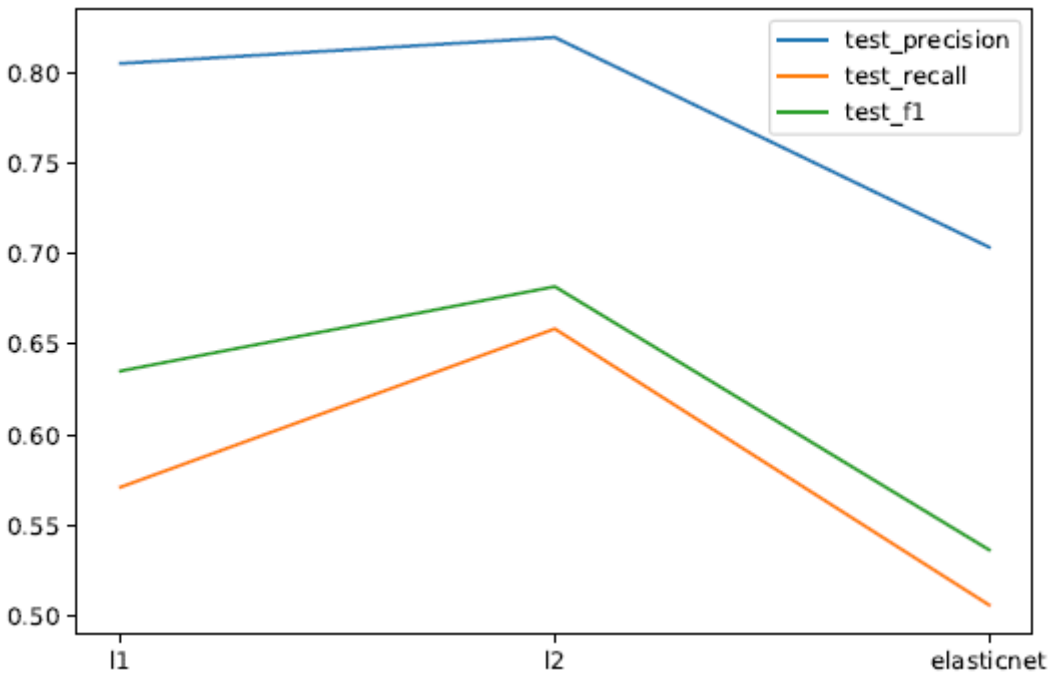


Рисунок 4.1 – Оцінки за штраф за регуляризацію

На рис.4.1 виконується перехресна перевірка, щоб знайти найкращий метод регуляризації. Результати показують, що регуляризація l2 є кращою.

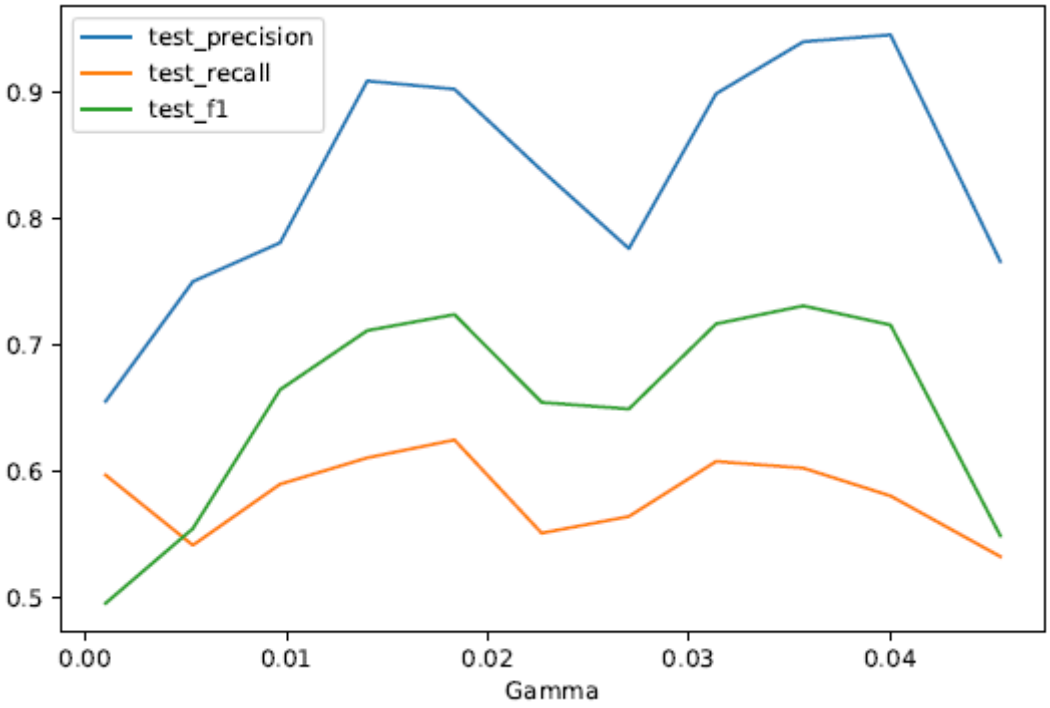


Рисунок 4.2 – Оцінки за Гамма

Для ядра радіальної базисної функції (RBF) на малюнку 34 показано, що найкращим параметром є γ= 0:03567.

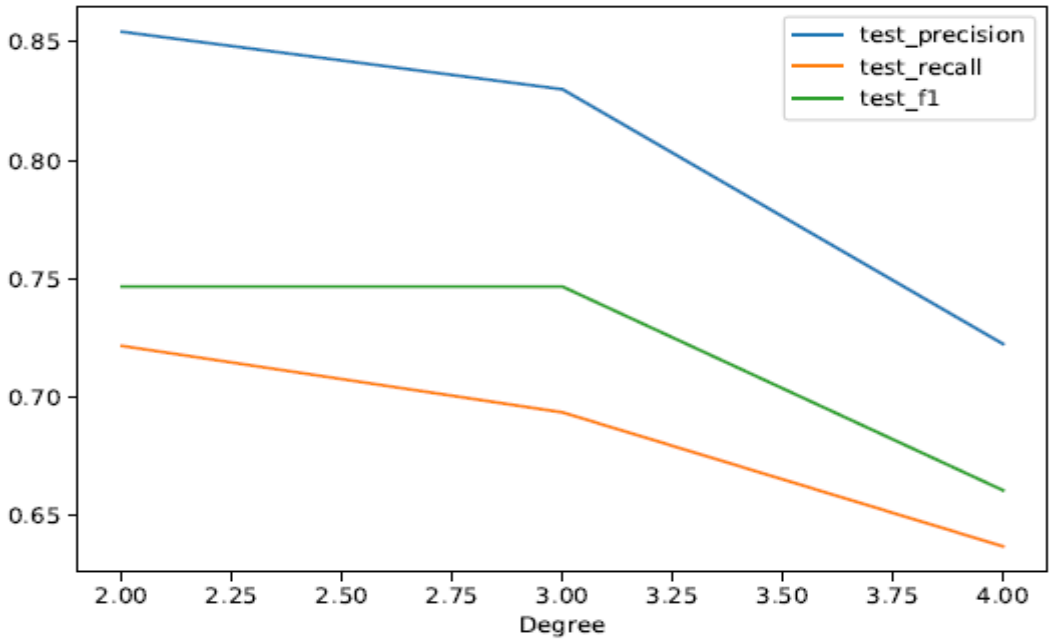


Рисунок 4.3 – Оцінки за поліноміальним ступенем

Для поліноміального ядра на рис. 4.3 показано, що найкращий ступінь для поліноміальної функції дорівнює 2.

Підводячи підсумок, найкращими параметрами SVM для виявлення ботнету в першому сценарії CTU-13 є поліноміальне ядро зі ступенем 2.

Випадковий ліс

Випадковий ліс — це алгоритм, який використовує кілька класифікаторів дерева рішень для прогнозування класу кожного вхідного потоку. Вибрана кількість дерев у лісі – 100.

Підвищення градієнта

Метод Gradient Boosting – це алгоритм, який також використовує класифікатори дерева рішень, але намагається поступово покращувати навчання, використовуючи оцінку одного дерева для побудови іншого. Потрібно налаштувати два основних параметри: втрату функції та максимальну глибину дерев (вибрана кількість дерев — 100).

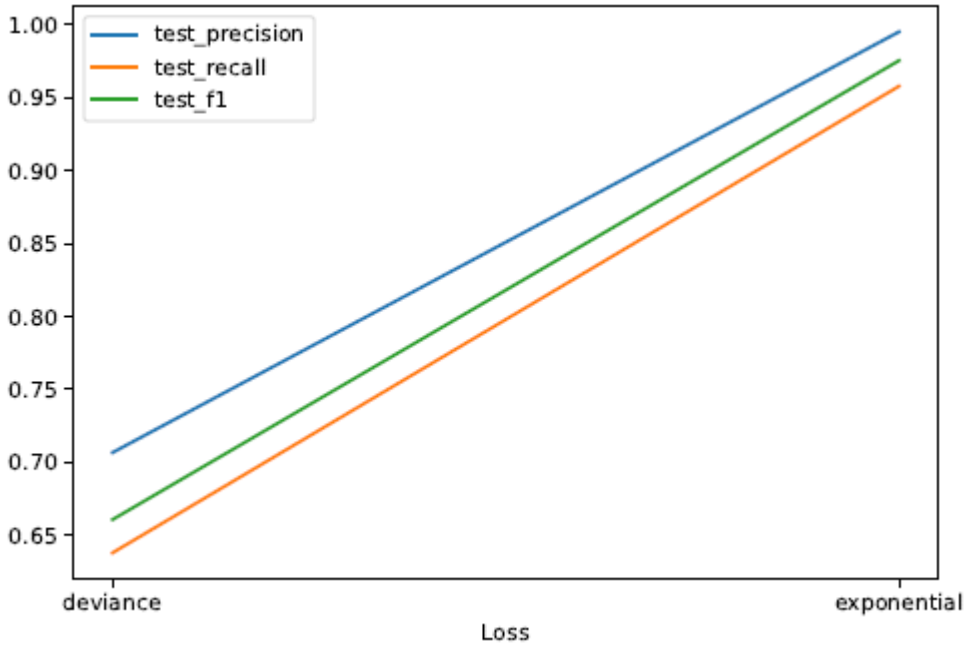


Рисунок 4.4 – Оцінки за вибрану програш

Рис.4.4 показує, що експоненційна втрата працює краще, ніж втрата від відхилення.

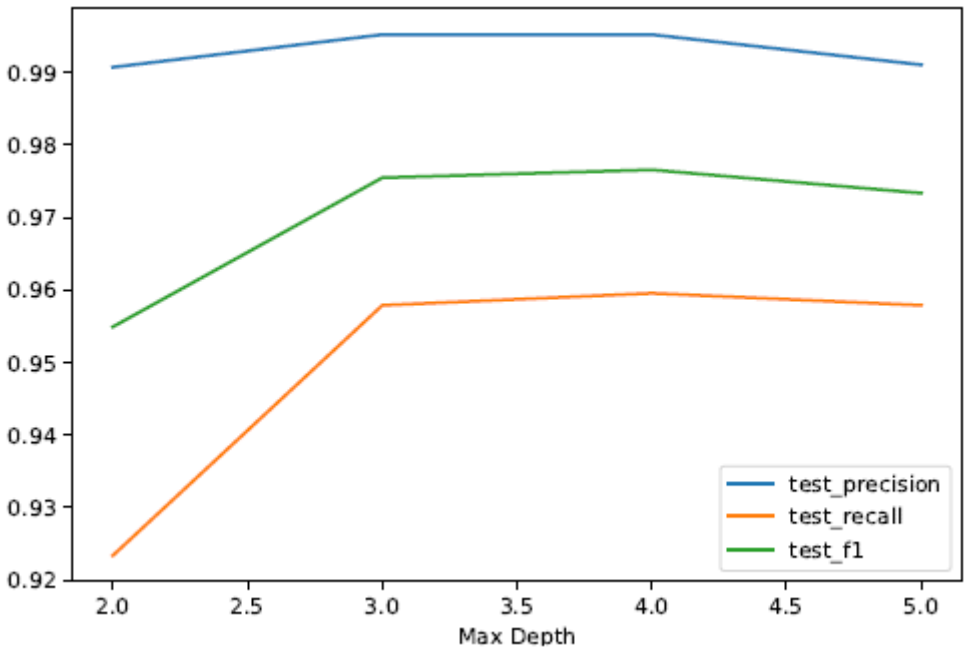


Рисунок 4.5 – Оцінки за максимальну глибину

На рис.4,5 показано, що найбільш релевантною максимальною глибиною для класифікатора є 4.

Щільна нейронна мережа

Останнім часом нейронні мережі набули популярності, оскільки вони дуже добре працюють, коли доступно багато даних.

Ми тестуємо тут просту щільну (або повністю підключену) нейронну мережу з 2 прихованими шарами: перший має 256 нейронів, а другий — 128.

Параметри нейронної мережі складаються з пакетної нормалізації, без випадання, функції активації ReLU (за винятком вихідного шару, де використовується сигмовидна функція). Модель має 39 681 параметри, які підлягають навчанню, і 768 параметрів, які не підлягають навчанню.

На малюнку 38 показані двійкові втрати перехресної ентропії для навчального набору та набору для перевірки (15% від усього набору) протягом 10 епох (використовується партія з 32).

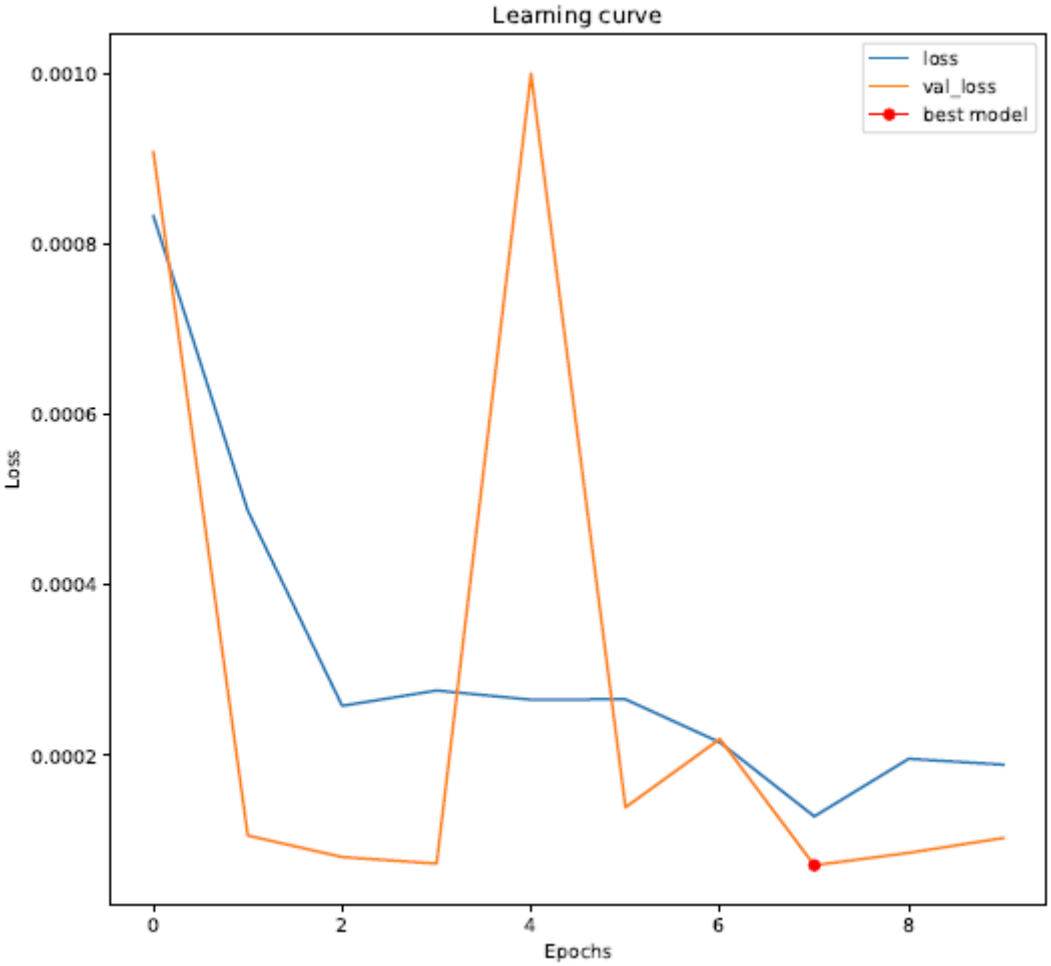


Рисунок 4.6 – Крива навчання нейронної мережі

Порівняння алгоритмів

Щоб порівняти алгоритми, ми навчаємо наші моделі з 2/3 набору даних (NetFlows вибираються випадковим чином) і тестуємо їх з рештою 1/3.

Таблиця 4.1 – Сценарій 1 Botnet Neris – підсумок результатів

Алгоритми	Тренування			Тест		
	Precision	Recall	f1 Score	Precision	Recall	f1 Score
Логістична регресія	0,74	0,97	0,83	0,74	0,96	0,84
Опорна векторна машина	0,94	0,80	0,86	0,91	0,78	0,86
Випадковий ліс	1,0	1,0	1,0	1,0	0,95	0,98

Підвищення градієнта	1,0	0,99	1,0	0,98	0,96	0,97
Щільна нейронна мережа	0,90	0,98	0,94	0,96	0,99	0,98

Таблиця 4.1 показує, що Random Forest, Gradient Boosting і Dense Neural Network перевершують інші протестовані алгоритми у виявленні бот-мереж серед мережевого трафіку з оцінкою f1 0,97. Тим не менш, методи логістичної регресії та машини опорного вектора успішно дають оцінку f1 0,85, один з низькою точністю, інший з низьким спогадом.

Оскільки Random Forest є найшвидшим алгоритмом для навчання і менш схильним до переобладнання (оскільки немає великої кількості гіперпараметрів для вибору), цей метод буде вивчено разом з іншими сценаріями в наступному розділі.

Виявлення різних бот-мереж

Щоб побачити, чи можна узагальнити результати Класифікатора випадкових лісів для сценарію 1 набору даних CTU-13, ми тестуємо його на всіх інших сценаріях.

Таблиця 4.2 – Підсумок результатів з використанням класифікатора випадкових лісів

Ботнет	Виборка		Тренування			Тест		
	К-ть	Ботнет, %	Precision	Recall	f1 Score	Precision	Recall	f1 Score
Neris - Scenario 1	2226720	0,128	1,0	1,0	1,0	1,0	0,95	0,975
Neris - Scenario 2	1431539	0,145	1,0	0,99	0,99	1,0	0,98	0,99
Rbot - Scenario 3	2024053	0,499	1,0	0,90	0,95	1,0	0,96	0,98
Rbot - Scenario 4	470663	0,236	1,0	0,90	0,95	1,0	0,69	0,82
Virut - Scenario 5	63643	0,346	1,0	1,0	1,0	1,0	0,25	0,4
DonBot - Scenario 6	220863	0,557	1,0	1,0	1,0	1,0	0,9	0,95

Sogou - Scenario 7	50629	0,138	1,0	1,0	1,0	1,0	0,25	0,4
Murlo - Scenario 8	1643574	0,68	1,0	1,0	1,0	1,0	0,94	0,97
Neris - Scenario 9	1168424	1,251	1,0	0,99	1,0	1,0	0,94	0,97
Rbot - Scenario 10	559194	0,967	1,0	0,98	0,99	1,0	0,90	0,95
Rbot - Scenario 11	61551	0,276	1,0	1,0	1,0	0,5	0,33	0,4
NSIS.ay - Scenario 12	156790	1,020	1,0	0,82	0,90	0,92	0,41	0,56
Virut - Scenario 13	1294025	0,757	1,0	1,0	1,0	1,0	0,96	0,98

Таблиця 4.2 показує, що методу класифікатор випадкових лісів вдається виявити більшість бот-мереж для 8 сценаріїв із 13. Погані оцінки 5 інших сценаріїв, схоже, пов’язані з розміром набору даних.

ВИСНОВКИ

В роботі було проведено розробку технології виявлення мережевих атак низької ефективності на основі аналізу трафіку по повному спектру параметрів з використання методів штучного інтелекту.

У результаті аналізу сценаріїв низькоінтенсивних атак було показано, що для аналізованих сценаріїв атак характерною є наявність певних пакетів, що повторюються з певною частотою за малою тимчасовою шкалою. Усі сценарії атак характеризуються такими основними особливостями:

- генерація невеликого за обсягом періодичного трафіку;
- однотипність елементів трафіку, що становлять атакуючу дію;
- неможливість віднести окремий запит або мережевий пакет до аномалій.

Використання запропонованої технології та програмного забезпечення написаного на його основі дозволить відвищити ефективність інструментів для виявлення та протидії атак.

Були протестовані різні алгоритми, серед яких логістична регресія, машина опорних векторів, випадковий ліс, градієнтний прискорення та щільна нейронна мережа. Класифікатор випадкових лісів був обраний для виявлення бот-мереж у всіх інших сценаріях, що призвело до точності виявлення понад 95% бот-мереж у 8 із 13 сценаріїв.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Charu C Aggarwal. 2017. Supervised outlier detection. In Outlier Analysis. Springer, 219–248.

2. Jinghui Chen, Saket Sathe, Charu Aggarwal, and Deepak Turaga. 2017. Outlier detection with autoencoder ensembles. In SDM. SIAM, 90–98.

3. François Chollet et al. 2015. Keras. https://keras.io. [4] Charles Elkan and Keith Noto. 2008. Learning classifiers from only positive and unlabeled data. In KDD. ACM, 213–220.

4. Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2012. Isolation-based anomaly detection. ACM Transactions on Knowledge Discovery from Data 6, 1 (2012), 3.

5. Thomas Schlegl, Philipp Seeböck, Sebastian M Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In IPMI. Springer, Cham, 146–157.

6. Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. In NeurIPS. 4077–4087.

7. Rohan Bapat et al. “Identifying malicious botnet traffic using logistic regression”. In: 2018 Systems and Information Engineering Design Symposium (SIEDS). IEEE. 2018, pp. 266–271.

8. B Claise. Trammell, B., Ed., and P. Aitken," Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. Tech. rep. STD 77, RFC 7011, September, 2013.

9. Melisha Dsouza. Build botnet detectors using machine learning algorithms in Python [Tutorial]. 2018. url: https://hub.packtpub.com/build-botnet-detectors-using-machinelearning-algorithms-in-python-tutorial/.

10. Levent Ertoz et al. The MINDS - Minnesota Intrusion Detection System. Ed. by Data Mining-Next Generation. 2004. Chap. 3.

11. P Fruehwirt, S Schrittwieser, and ER Weippl. “Using machine learning techniques for traffic classification and preliminary surveying of an attackers profile”. In: Proc. of Int. Conf. on Privacy, Security, Risk and Trust. 2014.

12. S. Garcia et al. “An empirical comparison of botnet detection methods”. In: ScienceDirect Computers & Security 45 (2014), pp. 100–123.

13. Jiangpan Hou et al. “Machine Learning Based DDos Detection Through NetFlow Analysis”. In: MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM). IEEE. 2018, pp. 1–6.

14. Rafal Kozik and Michal Choras. “Pattern Extraction Algorithm for NetFlow-Based Botnet Activities Detection”. In: Security and Communication Networks 2017 (2017).

15. Saakshi Kapoor, Vishal Gupta, and Rohit Kumar. “An Obfuscated Attack Detection Approach for Collaborative Recommender Systems”. In: Journal of Computing and Information Technology 26 (2018), pp. 45–56.

16. Zeinab Khorshidpour, Sattar Hashemi, and Ali Hamzeh. “Evaluation of random forest classifier in security domain”. In: Applied Intelligence 47 (2017), pp. 558–569.

17. Rafal Kozik, Marek Pawlicki, and Choras Michal. “Cost-Sensitive Distributed Machine Learning for NetFlow-Based Botnet Activity Detection”. In: Security and Communication Networks 2018 (2018).

18. Kozik Rafal and Michal Choras. “Machine Learning Techniques for Cyber Attacks Detection”. In: Image Processing and Communications Challenges 5 233 (2014), pp. 391–398.

19. Kozik Rafal, Michal Choras, and Jorg Keller. “Balanced Efficient Lifelong Learning (B-ELLA) for Cyber Attack Detection”. In: Journal of Universal Computer Science 25 (2019), pp. 2–15.

20. Martin Rehak et al. “CAMNEP: agent-based network intrusion detection system”. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track. International Foundation for Autonomous Agents and Multiagent Systems. 2008, pp. 133–136.

21. Cynthia Wagner, Jerome Francois, Thomas Engel, et al. “Machine learning approach for ip-flow record anomaly detection”. In: International Conference on Research in Networking. Springer. 2011, pp. 28–39.

22. Пришвидшений курс Python / Маттес Ерік; Видавництво Старого Лева, 2021. — 600 с. — ISBN 978-617-679-853-8.

ДОДАТОК А – ТЕКСТИ ПРОГРАМНИХ МОДУЛІВ НА PYTHON

```
import pandas as pd
import numpy as np
import datetime
import h5py

from scipy.stats import mode

window_width = 120 # seconds
window_stride = 60 # seconds

print("Import data")
data = pd.read_csv("capture20110812_3.binetflow")

print("Preprocessing")
def normalize_column(dt, column):
    mean = dt[column].mean()
    std = dt[column].std()
    print(mean, std)

    dt[column] = (dt[column]-mean) / std

data['StartTime'] = pd.to_datetime(data['StartTime']).astype(np.int64)*1e-9
datetime_start = data['StartTime'].min()

data['Window_lower'] = (data['StartTime']-datetime_start-
window_width)/window_stride+1
data['Window_lower'].clip(lower=0, inplace=True)
```

```
        data['Window_upper_excl'] = (data['StartTime']-
datetime_start)/window_stride+1
        data = data.astype({"Window_lower": int, "Window_upper_excl": int})
        data.drop('StartTime', axis=1, inplace=True)

        data['Label'], labels = pd.factorize(data['Label'].str.slice(0, 15))
        #print(data.dtypes)

        X = pd.DataFrame()
        nb_windows = data['Window_upper_excl'].max()
        print(nb_windows)

        for i in range(0, nb_windows):
            gb = data.loc[(data['Window_lower'] <= i) & (data['Window_upper_excl']
> i)].groupby('SrcAddr')
            X =
X.append(gb.size().to_frame(name='counts').join(gb.agg({'Sport':'nunique',
                                                    'DstAddr':'nunique',
                                                    'Dport':'nunique',
                                                    'Dur':['sum', 'mean', 'std', 'max', 'median'],
                                                    'TotBytes':['sum', 'mean', 'std', 'max',
'median'],
                                                    'SrcBytes':['sum', 'mean', 'std', 'max',
'median'],
                                                    'Label':lambda x:
mode(x)[0]})).reset_index().assign(window_id=i))
            print(X.shape)

        del(data)
```

```
X.columns = ["_".join(x) if isinstance(x, tuple) else x for x in X.columns.ravel()]
#print(X.columns.values)

# std can be Nan if only one element
X.fillna(-1, inplace=True)

#print(X.columns.values)
columns_to_normalize = list(X.columns.values)
columns_to_normalize.remove('SrcAddr')
columns_to_normalize.remove('Label_<lambda>')
columns_to_normalize.remove('window_id')

normalize_column(X, columns_to_normalize)

with pd.option_context('display.max_rows', 10, 'display.max_columns', 22):
    print(X.shape)
    print(X)
    print(X.dtypes)

#with pd.option_context('display.max_rows', 10, 'display.max_columns', 20):
#    print(X.loc[X['Label'] != 0])

X.drop('SrcAddr', axis=1).to_hdf('data_window_botnet3.h5', key="data", mode="w")
np.save("data_window_botnet3_id.npy", X['SrcAddr'])
np.save("data_window_botnet3_labels.npy", labels)

import numpy as np
```

```
import pandas as pd
from scipy.sparse import csc_matrix
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import h5py

from sklearn import model_selection, feature_selection, utils, ensemble, linear_model, metrics

print("Import data")

X = pd.read_hdf('data_window_botnet5.h5', key='data')
X.reset_index(drop=True, inplace=True)

X2 = pd.read_hdf('data_window3_botnet5.h5', key='data')
X2.reset_index(drop=True, inplace=True)

X = X.join(X2)

X.drop('window_id', axis=1, inplace=True)

y = X['Label_<lambda>']
X.drop('Label_<lambda>', axis=1, inplace=True)

labels = np.load("data_window_botnet5_labels.npy")

print(X.columns.values)
print(labels)
print(np.where(labels == 'flow=From-Botne')[0][0])
```

```
print("y", np.unique(y, return_counts=True))

y_bin6 = y==np.where(labels == 'flow=From-Botne')[0][0]

nb_prediction = 50
np.random.seed(seed=123456)
tab_seed = np.random.randint(0, 1000000000, nb_prediction)
print(tab_seed)

tab_train_precision = np.array([0.]*nb_prediction)
tab_train_recall = np.array([0.]*nb_prediction)
tab_train_fbeta_score = np.array([0.]*nb_prediction)

tab_test_precision = np.array([0.]*nb_prediction)
tab_test_recall = np.array([0.]*nb_prediction)
tab_test_fbeta_score = np.array([0.]*nb_prediction)

for i in range(0, nb_prediction):
    X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y_bin6, test_size=0.33, random_state=tab_seed[i])

    X_train_new, y_train_new = utils.resample(X_train, y_train, n_samples=X_train.shape[0]*30, random_state=tab_seed[i])

    print(i)
    print("y_train", np.unique(y_train_new, return_counts=True))
    print("y_test", np.unique(y_test, return_counts=True))

    clf = ensemble.RandomForestClassifier(n_estimators=100, random_state=tab_seed[i], verbose=0, class_weight=None)
```

```
            clf.fit(X_train_new, y_train_new)

            y_pred_train = clf.predict(X_train_new)
            precision, recall, fbeta_score, support = metrics.precision_recall_fscore_support(y_train_new, y_pred_train)
            tab_train_precision[i] = precision[1]
            tab_train_recall[i] = recall[1]
            tab_train_fbeta_score[i] = fbeta_score[1]

            y_pred_test = clf.predict(X_test)
            precision, recall, fbeta_score, support = metrics.precision_recall_fscore_support(y_test, y_pred_test)
            tab_test_precision[i] = precision[1]
            tab_test_recall[i] = recall[1]
            tab_test_fbeta_score[i] = fbeta_score[1]

        print("Train")
        print("precision = ", tab_train_precision.mean(), tab_train_precision.std(), tab_train_precision)
        print("recall = ", tab_train_recall.mean(), tab_train_recall.std(), tab_train_recall)
        print("fbeta_score = ", tab_train_fbeta_score.mean(), tab_train_fbeta_score.std(), tab_train_fbeta_score)

        print("Test")
        print("precision = ", tab_test_precision.mean(), tab_test_precision.std(), tab_test_precision)
        print("recall = ", tab_test_recall.mean(), tab_test_recall.std(), tab_test_recall)
        print("fbeta_score = ", tab_test_fbeta_score.mean(), tab_test_fbeta_score.std(), tab_test_fbeta_score)
```

```
import pandas as pd
import numpy as np
import datetime
import h5py

from scipy.stats import mode

window_width = 120 # seconds
window_stride = 60 # seconds

print("Import data")
data = pd.read_csv("capture20110812_3.binetflow")
#with pd.option_context('display.max_rows', None, 'display.max_columns',
15):
#    print(data.shape)
#    print(data.head())
#    print(data.dtypes)

print("Preprocessing")
def normalize_column(dt, column):
    mean = dt[column].mean()
    std = dt[column].std()
    print(mean, std)

    dt[column] = (dt[column]-mean) / std

data['StartTime'] = pd.to_datetime(data['StartTime']).astype(np.int64)*1e-9
datetime_start = data['StartTime'].min()
```

```
        data['Window_lower'] = (data['StartTime']-datetime_start-
window_width)/window_stride+1
        data['Window_lower'].clip(lower=0, inplace=True)
        data['Window_upper_excl'] = (data['StartTime']-
datetime_start)/window_stride+1
        data = data.astype({"Window_lower": int, "Window_upper_excl": int})
        data.drop('StartTime', axis=1, inplace=True)

        data['Label'], labels = pd.factorize(data['Label'].str.slice(0, 15))
        #print(data.dtypes)

        def RU(df):
            if df.shape[0] == 1:
                return 1.0
            else:
                proba = df.value_counts()/df.shape[0]
                h = proba*np.log10(proba)
                return -h.sum()/np.log10(df.shape[0])

        X = pd.DataFrame()
        nb_windows = data['Window_upper_excl'].max()
        print(nb_windows)

        for i in range(0, nb_windows):
            gb = data.loc[(data['Window_lower'] <= i) & (data['Window_upper_excl']
> i)].groupby('SrcAddr')
            X = X.append(gb.agg({'Sport':[RU],
                                 'DstAddr':[RU],
                                 'Dport':[RU]}).reset_index())
            print(X.shape)
```

```
del(data)

X.columns = ["_".join(x) if isinstance(x, tuple) else x for x in X.columns.ravel()]
#print(X.columns.values)

#print(X.columns.values)
columns_to_normalize = list(X.columns.values)
columns_to_normalize.remove('SrcAddr_')

normalize_column(X, columns_to_normalize)

with pd.option_context('display.max_rows', 10, 'display.max_columns', 22):
    print(X.shape)
    print(X)
    print(X.dtypes)

#with pd.option_context('display.max_rows', 10, 'display.max_columns', 20):
#    print(X.loc[X['Label'] != 0])

X.drop('SrcAddr_', axis=1).to_hdf('data_window3_botnet3.h5', key="data", mode="w")
np.save("data_window_botnet3_id3.npy", X['SrcAddr_'])
np.save("data_window_botnet3_labels3.npy", labels)
```