НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

До захисту допущено:

В.о.завідувача кафедри

_____ Оксана ТИМОЩУК

«___»_____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Системи та методи штучного інтелекту»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Аналіз мережевого трафіку»

Виконав (-ла):
студент (-ка) IV курсу, групи КА-65
Авксентьєва Іванна Олегівна _____

Керівник:
Асистент Кухарєв Сергій Олександрович _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О. А. _____
Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:
доцент, к.е.н. Безносик О.Ю. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 122 «Комп'ютерні науки та інформаційні технології»
Освітньо-професійна програма «Системи та методи штучного інтелекту»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Оксана ТИМОЩУК
«25» травня 2020 р.

ЗАВДАННЯ
на дипломну роботу студенту
Авксентьєвій Іванні Олегівні

1. Тема роботи «Аналіз мережевого трафіку», керівник роботи Кухарєв Сергій Олександрович, асистент, затверджені наказом по університету від «25» травня 2020 р. № 1143-с

2. Термін подання студентом роботи 8.06.2020.

3. Вихідні дані до роботи

Вихідними даними є методичні рекомендації, вимоги до параметрів аналізу мережевого трафіку.

Вхідними даними для проведення аналізу є існуючі пакетні сніфери та їх характеристики.

4. Зміст роботи

1. Ознайомитись з засобами моніторингу мережевого трафіку

2. Визначити критерії аналізу мережевого трафіку.

3. Провести аналіз за обраними критеріями та зробити висновки.

4. Розробити програмне забезпечення згідно виконаного аналізу.

5. Перелік ілюстративного матеріалу: тенденції росту мережевого трафіку, порівняння існуючих аналізаторів, FATT(Fingerprint all the things), фази моделювання мережевого трафіку, висновки.

5.1. Презентація до захисту роботи.

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А., доцент		

7. Дата видачі завдання 01.02.2020_____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	01.02.2020	Виконано
2	Збір інформації	15.02.2020	Виконано
3	Аналіз вимог завдання, вибір методів і засобів розв’язання поставленої задачі	15.03.2020	Виконано
4	Провести пошук та ознайомитися з існуючими аналізаторами мережевого трафіку	25.03.2020	Виконано
5	Проведення порівняльного аналізу аналізаторів мережевого трафіку	10.04.2020	Виконано
6	Реалізація базового функціоналу аналізатора мережевого трафіку	25.04.2020	Виконано
7	Моделювання мережевого трафіку	07.05.2020	Виконано
8	Оформлення дипломної роботи	31.05.2020	Виконано
9	Отримання допуску до захисту та подача роботи в ДЕК	02.06.2020	Виконано

Студент Авксентьєва І.О.
Керівник Кухарєв С.О.

* Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота містить: 105 с., 4 ч., 17 табл., 23 рис., 10 дод., 28 джерел.

Об’єкт дослідження - аналізатори та генератори мережевого трафіку.

Мета роботи - проведення порівняльного аналізу існуючих засобів моніторингу мережевого трафіку, створення власниого аналізатор з базовим функіоналом, дослідження методів моделювання мережевого трафіку.

Методи дослідження - порівняльний аналіз, моделювання мережевого трафіку.

В роботі було розглянуто сніферні пакети Wireshark, TCPDump, Colasoft за критеріями:

1. Користувацького інтерфейсу;
2. Кросплатформності;
3. Відкритості коду;
4. Вартості;
5. Розміром на диску;
6. Виявлення нетипових та ненормальних даних;
7. Можливістю розробки та налаштування розробниками;
8. Можливістю підтривувати різні протоколи.

Також проведено розробку власного ПЗ на основі анадізу існуючяих сніферів. Розглянуто теоретичну частину генерації мережевого трафіку. Виконано аналіз та зазначено можливі напрямки майбутнього дослідження алгоритму генерації трафіку.

МЕРЕЖЕВИЙ ТРАФІК, АНАЛІЗАТОРИ, СНІФЕРИ, АНАЛІЗ, ГЕНЕРАЦІЯ.

ABSTRACT

Report: 105 p., 4 p., 17 tables, 23 figures, 10 appendices, 28 sources.

The object of research - analyzers and generators of network traffic.

The purpose of this work is to conduct a comparative analysis of existing tools for monitoring network traffic, creating your own analyzer with basic functionality, research methods for modeling network traffic.

Research methods - comparative analysis, modeling of network traffic.

The sniffer packages Wireshark, TCPDump, Colasoft were considered in the work according to the criteria:

(1)User interface;

(2)Cross-platform;

(3)Code openness;

(4)Costs;

(5)Disk size;

(6)Detection of atypical and abnormal data;

(7)Ability to develop and customize developers;

(8)Ability to support different protocols.

We also developed our own software based on the analysis of existing sniffers. The theoretical part of network traffic generation is considered. The analysis is performed and the possible directions of the future research of the traffic generation algorithm are indicated.

NETWORK TRAFFIC, ANALYZERS,SNIFFERS, ANALISIS, GENERATION.

ЗМІСТ

Перелік умовних позначень, символів, скорочень і термінів 8
ВСТУП.....9
1. Аналіз та моніторинг мережевого трафіку 11
1.1.Аналіз мережевого трафіку: управління через АРМ.....11
1.2.«Глибина» аналізу мережевого трафіку 15
1.3.Особливості передачі мережевого трафіку 19
1.4.Висновки 23
2. Засоби моніторингу мережевого трафіку.....25
2.1.Пакетні сніфери 25
2.2.Огляд інструментів обробки пакетів 28
2.3.Порівняльний аналіз інструментів обробки пакетів 35
2.4.FATT(Fingerprint all the things) 42
2.5.Висновки 46
3. Моделювання мережевого трафіку.....47
3.1.Огляд генераторів мережевого трафіку.....47
3.2.Формування ідеї.....48
3.3.Прототип генератора мережевого трафіку.....50
3.4.Методи оцінки генератора трафіку.....58
3.5.Оцінка генератора трафіку 60
3.6.Висновки 65
4. Функціонально-вартісний аналіз 66
4.1.Постановка задачі проектування 66
4.2.Обґрунтування функцій та параметрів програмного продукту ..66
4.3.Економічний аналіз варіантів розробки ПП. 71
4.4.Обґрунтування функцій та параметрів програмного продукту ..76
4.5.Висновки 76
Висновки 78
Перелік посилань.....80
Додаток А НАУКОВІ ДОСЯГНЕННЯ 83
Додаток Б ЛІСТИНГ ПРОГРАМИ 84
Додаток В РЕЗУЛЬТАТИ РОБОТИ.....88

Додаток Г РЕЗУЛЬТАТИ РОБОТИ (JSON FORMAT).....89
Додаток Д ТЕСТУВАННЯ ВРАЗЛИВОСТЕЙ CVE-2020-0708 RDP.....93
Додаток Е ТЕСТУВАННЯ ВРАЗЛИВОСТЕЙ CVE-2020-0708 POC.....96
Додаток Є ДЕКОДУВАННЯ.....97
Додаток Ж МОДЕЛЬ МАРКОВА.....98
Додаток З МОДЕЛЬ НЕЙРОННОЇ МОВИ 99
Додаток И ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ 103

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

API - Application Programming Interface

APM - Application Performance Management

HTTP - HyperText Transfer Protocol

OSI - The Open Systems Interconnection model

RDP - Remote Desktop Protocol

SSL - Secure Sockets Layer

SSH - Secure SHell

TLS - Transport Layer Security

ВСТУП

Активний розвиток інформаійних технологій робить їх невідємною частиною сьогодення. Інформаційні системи сьогодні широко експлуатуються як і в комерційних, так і в державних. Їх взаємодія між собою відбувається через мережі.

Спостерігається стрімкий зріст об'єму мережевого трафіку, що провокує ускладнення його структури. Аналіз трафіку стає все більш потрібним в областях контроля та управління, оптимізації, захисту від шкідливого втручання. Моніторинг трафіку також необхідний, щоб більш ефективно діагностувати і вирішувати проблеми, коли вони постають, таким чином не доводячи різні мережеві сервіси до простоя впродовж тривалого часу. Доступно багато різних інструментів, які дозволяють допомогти адміністраторам з моніторингом та аналізом мережевого трафіку. Дана робота висвітлює методи моніторингу мережевого трафіку, надає порівняльну хавактеристику засобів мережевого трафіку, та досліджує методи моделювання мережевого трафіку

Моніторинг мережі — складне завдання, що вимагає великих сил та витрат, яке є невід’ємною складовою повсякденного життя мережевих адміністраторів. Адміністратори постійно прагнуть підтримати безперебійну роботу своєї мережі. Якщо мережа «впаде» хоча б на невеликий період часу, продуктивність в компанії скоротиться і (в разі організацій які надають державні послуги) сама можливість надання основних послуг буде поставлена під загрозу. У зв'язку з цим адміністраторам необхідно проводити постійний аналіз мережевого трафіку, стежити за його рухом і продуктивністю на всій мережі і перевіряти, чи з'явилися в ній проломи в безпеці. В випадках моніторингу при виявленні проблеми після її вирішення постає питання тестування коректності та цілісної роботи системи для цього в допомогу приходить моделювання трафіка, що дозволяє змоделювати трафік для здійснення аналізу, щоб зробити висновки можливих вразливостей системи, її

продуктивності, досліджені різних протоколів, алгоритмів або топографій мереж. Це обумовлює актуальність обраної теми.

Метою данної роботи є проведення порівняльного аналізу існуючих засобів моніторингу мережевого трафіку, створення власниого аналізатор з базовим функіоналом, дослідження методів моделювання мережевого трафіку.

Результатами данної роботи є власний аналізатор трафіку з базовим функіоналом на основі порівняльного аналізу та алгоритм моделювання мережевого трафіку зі збереженням послідовності.

1. АНАЛІЗ ТА МОНІТОРИНГ МЕРЕЖЕВОГО ТРАФІКУ

Аналіз мережевого трафіку набуває все більшу актуальність у зв’язку з розвитком мережевих технологій, збільшенням обсягу даних, переданих по мережі, впровадженням великої кількості нових мережевих протоколів (в тому числі закритих). В якості основних областей практичного застосування можна виділити наступні:

- виявлення проблем в роботі мережі;
- тестування(налагодження мережевих протоколів);
- запобігання мережевих атак;
- класифікація трафіку.

Поглиблений аналіз трафіку в даний час є однією з основних технологій при вирішенні практичних завдань пов'язаних із забезпеченням безпеки мереж, оптимізацією пропускної здатності каналів передачі даних, контролем якості зв'язку та інших. Якісне рішення згаданих завдань вимагає максимальної глибини розбору мережевого трафіку і надання результатів розбору в зручній формі. [12]

Можливість аналізу трафіку на рівні контенту додатків є у вузького кола закритих комерційних систем таких фірм, як Checkpoint і Cisco. Ці системи не дозволяють користувачам вільно розширювати перелік підтримуваних мережевих протоколів, їх закритість ускладнює інтеграцію. У той же час існуючі вільно поширювані інструменти не забезпечують належної якості проведеного аналізу, внаслідок чого з їх допомогою не завжди вдається отримати дані протоколів прикладного рівня.

1.1.Аналіз мережевого трафіку: управління через АРМ

Аналіз трафіку — необхідна область для управління, контролю і оптимізації, а також для захисту систем. Сучасні корпоративні мережі стають все більш складними завдяки додаванню численних персональних пристроїв, безлічі хмарних додатків, точок доступу, віртуальних серверів

тощо. Команди адміністраторів повинні ретельно керувати пропускною здатністю та якістю обслуговування, впроваджувати балансири навантажень для високої доступності, створювати надмірне середовище для резервного копіювання та забезпечувати синхронізовану роботу декількох компонентів у гібридному середовищі.

Аналіз можна здіснювати на рівні окремого пакету, в контексті роботи додатку або в діагностиці складних систем.

Для якісного управління додатком розберемо, що собою являє APM(Application Performance Management).

APM — це процес, який забезпечує зв’язок між роботою програми та всієї структури, в яку додаток входить.Також виміряє, оцінює і документує основні зв’язки додатку. Тобто в додатку ми повинні не лише підраховувати час роботи або використанний трафік, а у разі помилок системи повинні знайти їх причину та знайти методи їх усунення.

Також необхідно враховувати, що всі процеси повинні відбуватись одночасно для мережевого обладнання, каналів зв’язку, серверів тощо.

1.1.1.Види моніторингу APM

На базі процесу APM можна виділити види моніторингу:

1. На основі метрик додатків -кілька інструментів використовують різні показники сервера та додатків і називають це APM. У кращому випадку вони можуть повідомити, скільки запитів отримує ваша програма та потенційно, які URL-адреси можуть бути повільними. Оскільки вони не займаються профілюванням рівня коду, вони не можуть сказати вам, чому.

2. Ефективність на рівні коду - Stackify Retrace, New Relic, AppDynamics і Dynatrace - типовий тип APM, на основі кодування профілю та відстеження транзакцій.

3. На основі мережі - Extrahop використовує термін APM стосовно їх здатності вимірювати продуктивність додатків на основі мережевого

трафіку. Існує ціла категорія товарів під назвою NPM, яка зосереджена на цьому типі рішень.

1.1.2.Критичні особливості APM

Ось деякі ключових особливостей, яких притримуються більшість розробників при обробці данних:

1. Виконання кожного веб-запиту та транзакції - в основі APM ви маєте змогу оцінювати ефективність кожного веб-запиту та транзакції у вашій програмі. Потім ви можете скористатися цим, щоб зрозуміти, до яких запитів програма звертається найбільше, які найповільніші та які саме слід додати для покращення роботи.

2. Профілювання продуктивності на рівні коду - якщо ви хочете зрозуміти, чому ваша програма повільна, викидає помилки або є в ній дивні помилки, вам доведеться перейти до рівня коду. Знаючи, що певний веб-запит не працює, важливо з'ясувати, чому, іноді це дуже складно.

3. Використання та ефективність усіх залежностей програми, таких як бази даних, веб-сервіси, кешування тощо - чому ваш додаток повільний, зазвичай зводиться до швидкого трафіку або до проблеми з однією із ваших програм.Дуже часто зустрічаються такі проблеми:

- Конкретний запит SQL відбувається повільно
- Сервер баз даних SQL не працює
- Сусіди в хмарі викликають проблеми

4. Деталізація окремих веб-запитів чи транзакцій- вирішувати проблеми на виробництві дуже важко. Сліди транзакцій значно полегшують цю роботу, оскільки ви можете бачити деталі про те, що саме відбувається у вашому коді та як це впливає на ваших користувачів.

5. Основний моніторинг сервера та показники, такі як процесор, пам'ять тощо -проблеми із додатком можуть виникати з багатьох причин. Завдяки віртуалізації та хмарі сервер, який опускається, не є настільки поширеним в ці дні. Однак це все ж таки відбувається і це те, за чим

потрібно стежити. Також важливо стежити за такими речами, як CPU сервера та пам'ять. Багато сучасних веб-додатків зазвичай не пов'язані з CPU, але вони все ще можуть використовувати багато CPU, і це корисний показник для автоматичного масштабування вашої програми в хмарі.

6. Показники рамки програми, такі як лічильники продуктивності, JMX MBeans тощо - такі показники сервера, як CPU та пам'ять, цікаві, але для розробників показники додатків можуть бути набагато більш цінними для моніторингу справжньої роботи програми. Розробникам необхідно відстежувати показники навколо таких предметів, як збирання сміття, чергування запитів, обсягів транзакцій, часу завантаження сторінки та багато іншого. Також вони можуть стежити за різноманітними лічильниками продуктивності Windows та JMX MBeans. Це також може бути критично важливим для моніторингу таких речей, як Redis, Elasticsearch, SQL та інших сервісів за ключовими показниками.

7. Показники спеціальних програм, створені командою розробників або бізнесом - стандартні показники сервера та додатків можуть бути дуже корисними для моніторингу ваших програм. Однак ви можете отримати набагато більшу цінність, створивши та відстежуючи власні спеціальні показники.

8. Дані журналу додатків-дані журналу - це важлива частина роботи розробників для відслідковування запитів, коли їх програми розгорнуті. Розробникам потрібен доступ до своїх журналів через централізоване рішення для ведення журналів, як продукт управління журналом. На щастя, управління журналом - це включена функція APM в Retrace.

9. Помилки програми-як розробники, нам потрібно знати, коли у користувача відбувається помилка на сторінці, і важливо також знати, яка саме. Помилки - це перша лінія захисту для пошуку проблем із застосуванням. [4]Нам потрібно знайти та виправити помилки або принаймні знати про них, перш ніж клієнти повідомлять нам про це самі, або перестануть бути нашими користувачами.

Відмінне відстеження помилок, звітування та оповіщення є надзвичайно важливими для розробників в системі управління продуктивністю додатків. Я настійно рекомендую налаштувати сповіщення про нові винятки, а також для моніторингу загальної кількості помилок. Швидше за все, ви знайдете якісь нові помилки, які зможете швидко визначити та виправити.

10. Реальний моніторинг користувачів (RUM) - зозуміння продуктивності ваших програм на стороні сервера важливо. Однак сьогоднішні програми використовують стільки javascript, що важливо також відстежувати, скільки часу їх браузеру потрібно повністю завантажувати та відтворювати ваші веб-сторінки. Проста помилка JavaScript або повільне завантаження файлу javascript може повністю зіпсувати вашу програму. Справжній моніторинг користувачів або RUM - ще одна важлива особливість APM, що розробникам потрібно повністю контролювати свої програми.

1.2.«Глибина» аналізу мережевого трафіку

Розглядаються мережі з комутацією пакетів. Рішення практичних завдань аналізу полягає в розборі заголовків мережевих протоколів в пакетах і відновленні потоків даних, що передаються. Можна виділити 3 группи при проведення аналізу трафіку(рисунок 1.1):

- поверхневий (SPI — Shallow Packet Inspection);
- помірний (MPI — Medium Packet Inspection);
- поглиблений (DPI — Deep Packet Inspection).

1.2.1.Поверхневий аналіз пакетів(SPI)

Технологія аналізу трафіку, яка грунтується виключно на заголовках пакету рівнів L1-L3 по моделі OSI. Потребує низькі вимоги до обчислювальних ресурсів, що дозволяє аналізувати великі обсяги трафіку. Технологія широко поширена, на її основі працює більшість міжмережевих

екранів операційних систем (зокрема в ОС Windows XP / Vista і OS X), маршрутизаторів і інших мережевих пристроїв.[11] На її основі реалізовані мережеві списки контролю доступу на рівні IP адрес і портів (Access Control List, ACL). Таким чином, дана технологія добре підходить для розмежування доступу ззовні до окремих комп'ютерів (IP) і сервісів (порти) внутрішньої мережі.

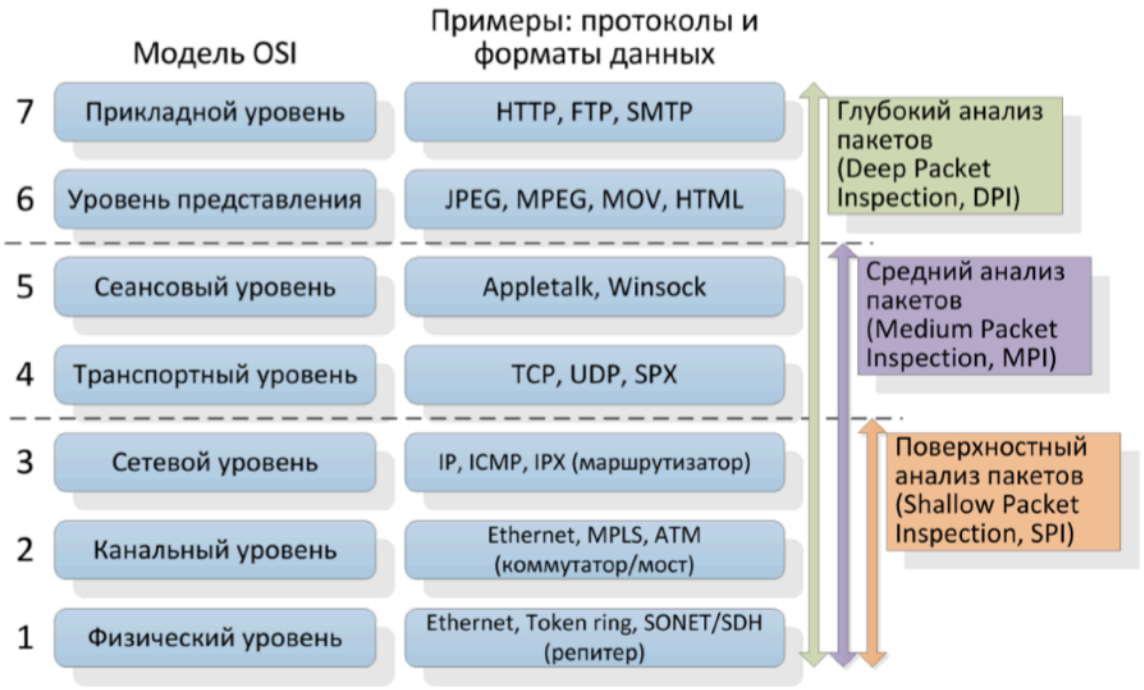


Рисунок 1.1. Відповідність рівнів моделі «OSI» «глибині» аналізу мережевого трафіку

1.2.2.Помірний аналіз пакетів(MPI)

Технологія аналізу трафіку, яка грунтується на інспектуванні сесій і сеансів зв'язку, ініційованих додатком, але встановлюваних шлюзом-посередником (див. Рисунок 1.2.). Також застосовується термін «проксі додатків» (application proxy). В рамках даної технології вміст пакетів аналізується частково і по визначеним правилам. Не використовуються складні методи аналізу типу сигнатурного. Пристрої, що реалізують даний функціонал розміщуються між провайдером інтернету і кінцевим користувачем. Дані пристрої розбирають заголовки аж до транспортного

рівня і невелику частину даних пакета для зіставлення розібраної частини з деяким списком розбору (parse list), з подальшою реакцією в разі їх виявлення. Дані списки зазвичай коротше списків ACL і надають більш широкий діапазон дій на відміну від «дозволити / заборонити» в разі ACL. Ці списки також більш виразні, так як дозволяють прив'язуватися не до IP-адреса, а до формату даних пакетів і даними деяких протоколів рівня додатки, наприклад, URL-адресів в разі протоколу HTTP. За допомогою MPI можна, наприклад, заблокувати можливість отримання flash-файлів або картинок з певних інтернет сервісів (на рівні уявлення OSI) або заблокувати частину команд (на рівні додатку OSI) в окремих протоколах. Набір протоколів, як правило, дуже обмежений. Наприклад, в перших версіях CheckPoint FireWall-1 (CheckPoint FW-1) підтримувалися протоколи Telnet, FTP, HTTP, а в Cisco Private Internet Exchange (Cisco PIX) - FTP, HTTP, H.323, RSH, SMTP і SQLNET.[13] Згодом дані набори незначно розширювалися. Також відомо, що дана технологія використовується в продуктах компаній McAfee і Symantec. Міжмережеві екрани, що використовують цю технологію, відносяться до другого покоління. Дана технологія більш гнучка в порівнянні з SPI і, крім розмежування доступу, підходить для більшого числа завдань - кешування вмісту, аналіз стисненого / шифрованого трафіку, обмеження функціонала окремих протоколів шляхом заборони окремих команд. Завдяки підключенню в режимі проксі, може служити в якості Wan Optimizer'а.

Основний недолік MPI - погана масштабованість: кожна команда і протокол вимагають окремого «шлюзу» (вхідний-вихідний порти). Крім того, робота в режимі проксі сильно знижує швидкість обробки. Для зниження навантаження на проксі-сервер був розроблений протокол ICAP, що дозволяє проксі-серверам відправляти через них дані для проведення аналізу стороннім серверам на предмет безпеки або аналізу вмісту. Ця схема реалізована в антивірусному продукті ClamAV, який може підключатися до проксі-серверів Squid і NetCache. [19] Ці фактори сильно

обмежують застосування даної технології на рівні провайдерів інтернету внаслідок необхідності аналізу великого числа протоколів і команд на широких каналах зв'язку.

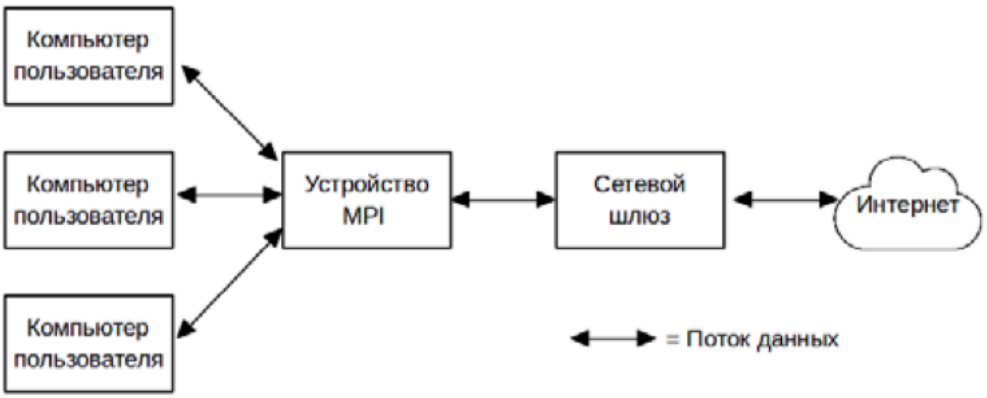


Рисунок 1.2. Схема застосування пристроїв аналізу на основі технології MPI.

1.2.3.Поглиблений аналіз пакетів(DPI)

Іноді вживають більш вузький термін - DPP (Deep Packet Processing), який має на увазі такі дії над пакетами, як модифікація, фільтрація або перенаправлення. Сьогодні обидва терміни часто використовуються як взаємозамінні . Дана технологія є логічним розвитком MPI. В рамках даного підходу аналізатор переглядає вміст кожного пакета повністю. Одним з важливих відмінностей від попередніх технологій є те, що системи на базі DPI можуть приймати рішення не тільки по вмісту пакетів, але і за непрямими ознаками, властивим якимось певним мережевим програмами і протоколам. Для цього може використовуватися статистичний аналіз.

Наприклад, аналіз частоти зустрічі певних символів, довжин пакетів, відстань між мітками часу послідовних пакетів і т.д. Також, в порівнянні з попередніми підходами, значно розширено список застосувань технології: класифікація, обмеження смуги, пріоритезація, маркування, кешування і т. Д. Технологія DPI отримала розвиток, перш за все, через стрімке зростання обчислювальних здібностей процесорів, їх швидкодії і, відповідно, можливостей для більш повного і точного аналізу мережевих даних.[19]

На відміну від MPI, дана технологія спочатку розроблялася для високошвидкісної обробки та ідентифікації великої кількості додатків в реальному часі. Таким чином, рішення на основі DPI добре масштабуються як по ширині мережевого каналу (відомі рішення, що працюють на каналах близько 100 Гбіт / сек), так і за кількістю ідентифікованих додатків (в існуючих рішеннях - порядку декількох тисяч). З точки зору реалізації, основний компонент будь-якого рішення DPI - модуль класифікації, що відповідає за класифікацію мережевих потоків. При цьому в залежності від цілей застосування DPI, класифікація може виконуватися з різною точністю:

- тип протоколу або додатка (наприклад, Web, P2P, VoIP);
- конкретний протокол рівня додатка (HTTP, BitTorrent, SIP);
- додаток, що використовує протокол (Google Chrome, µTorrent,Skype).

Важливо відзначити, що відповідність між класами різних рівнів точності неоднозначно, що показано на Рисунку 1.3.

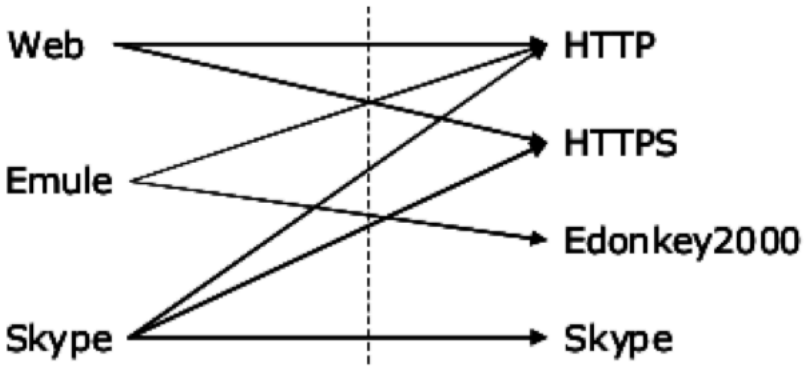


Рисунок 1.3. Різниця між ідентифікацією додатків (зліва) і протоколів (праворуч).

1.3.Особливості передачі мережевого трафіку

Кожен мережевий пакет складається з керуючої інформації і корисного навантаження. Тут і далі термін «пакет» застосовується в якості універсального для узагальнення таких понять, як фрейм, дейтаграмма,

сегмент відповідних мережевих протоколів. В процесі розбору в пакеті виділяються заголовки протоколів, аналізуються значення полів в них.

Структура заголовка визначається специфікацією, тоді як корисне навантаження може містити довільним чином організовані дані, хоча зазвичай являє собою пакет протоколу наступного, більш високого рівня: для продовження розбору необхідно визначати, який це протокол (Рисунок 1.4.).

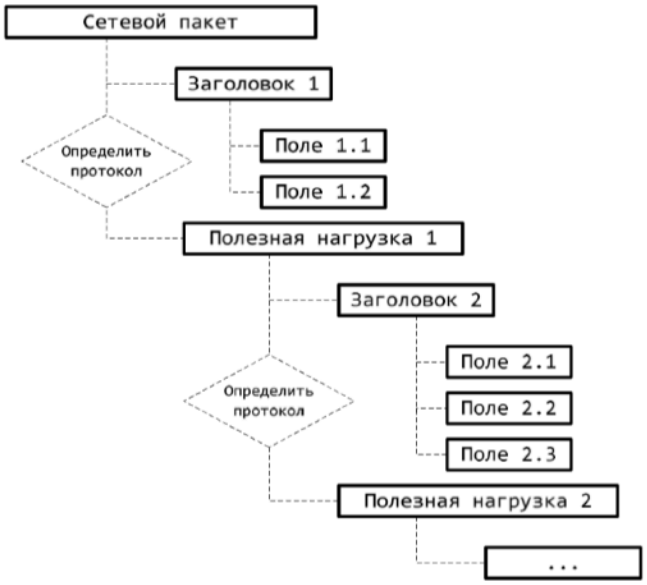


Рисунок 1.4. Виділення та розбір заголовків протоколів в пакеті

Відповідно до моделі OSI заголовки мережевих протоколів пакета утворюють стек і, як правило, слідують один за одним у природному порядку - від низького рівня до високого. Однак при організації тунельних з'єднань цей порядок може бути порушений - наприклад, при передачі IPv4-пакетів (мережевий рівень) в рамках пакетів протоколу UDP [20] (транспортний рівень). Тунельні протоколи в даний час набули широкого поширення: зокрема, вони використовуються при організації віртуальних приватних мереж. [17] У загальному випадку можлива побудова тунелю довільної конфігурації: зокрема, один тунель може бути вкладений в інший. Розбір тунельного трафіку повинен підтримуватися мережевим аналізатором.

Виділяють протоколи зі збереженням і без збереження стану. Обов'язковою частиною специфікації протоколу зі збереженням стану є відповідний автомат станів (Protocol State Machine). При проведенні аналізу в режимі реального часу кількість з'єднань, для яких необхідне збереження характеристик поточного стану, може необмежено рости. Тому аналізатор повинен гнучко управляти розподілом доступних йому ресурсів.

Важливою характеристикою мережевого протоколу є MTU (Maximum Transmission Unit) - максимальний розмір даних, які можуть бути передані в рамках одного пакету. Для протоколу IPv4 значення MTU складає 65535. Оскільки на практиці IPv4-пакети зазвичай инкапсулируются в Ethernet-фрейми, результуюче значення MTU визначається відповідно до конкретної версією стандарту Ethernet [21], підтримуваного мережевим обладнанням. Для блоків даних з розміром, що перевищує MTU, проводиться фрагментація: відправник розбиває блок на порції допустимого розміру, після чого кожна порція передається в рамках окремого пакета. Одержувач, таким чином, повинен виконати дефрагментацію: відновити вихідний блок з отриманих окремо порцій. Для протоколу IPv4 останній фрагмент визначається скинутим прапором MF (More Fragments): при цьому в ньому не містяться (Рисунок 1.5.) дані наступної PDU (Protocol Data Unit - одиниця передачі).

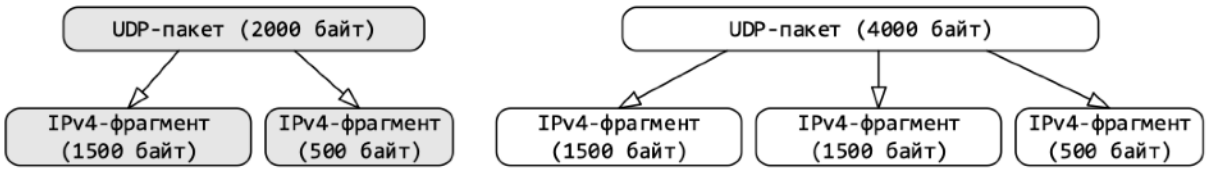


Рисунок 1.5. Приклад фрагментації IPv4

У випадку протоколу TCP (Рисунок 1.6.) неформальною ознакою «останнього» для заданого PDU сегмента є PSH-прапор, однак цей сегмент в загальному випадку містить дані наступної одиниці передачі звідси виникає завдання визначення меж.

Рисунок 1.6. Приклад сегментації TCP

Таким чином для проведення глибокого аналізу потрібно:

- відновлювати початковий порядок пакетів;
- визначати грані вище за списком PDU.

Питання з упорядкування пакетів для протоколів IP та TCP розглядаються в [22, 23].

Для забезпечення безпеки з'єднань деякі протоколи передбачають передачу даних в зашифрованому виді наприклад сімейство протоколів TLS [24, 25]. Щоб проаналізувати зашифровані дані, необхідно до того їх розшифрувати, використав наданий користувачем ключ(Рисунок 1.7.): аналізатор повинен мати інтерфейс для додавання відсутньої для проведення розбору інформаії.

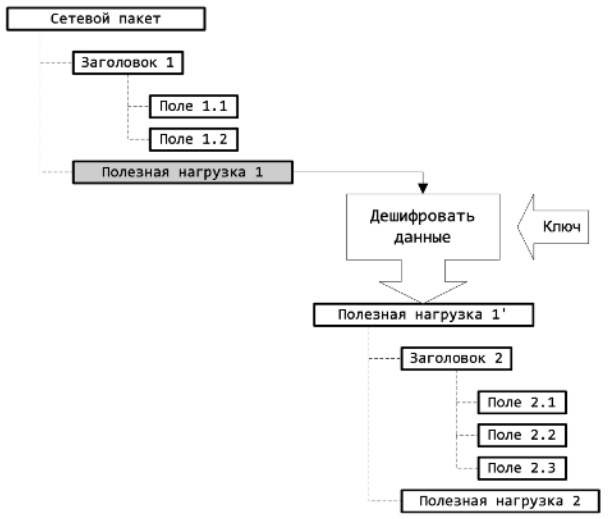


Рисунок 1.7. Розшифровка даних з використанням зовнішнього ключа

При аналізі трафіку неминуче виникають помилки розбору. Під цією помилкою розуміється невідповідність між спеифікаією протоколу(кодом який виконує розбір) та даними, розбір яких проводиться згідно цієї специфікаії. Причини винекнення помилок розбору різняться:

- недокументовані можливості протоколу;
- викривлення данних при транспортуванні мережею;
- помилки в коді аналізатора;

Помилки розбору повинні легко локалізуватися та відтворюватися. Якщо помилка, яка виникла не є критичною, аналіз повенен продовжуватися.

1.4.Висновки

Аналізатори мережевого трафіку, як правило, мають модульну архітектуру [26]: з часом з'являються нові протоколи, і їх необхідно підтримувати. Розширювати систему, в якій функції розбору даних всіх протоколів зосереджені в одному функціональному модулі, важко. У разі модульної архітектури для кожного протоколу створюється окремий модуль, в якому визначаються методи та структури даних для роботи з цим протоколом. Виникає додатковий питання про залежності: при додаванні нового модуля необхідно «повідомити» про його існування іншим. Вносити зміни в код існуючих модулів неефективно: може бути порушена логіка їх роботи або внесені помилки, налагодження яких важке. До того ж потрібна повторна збірка змінених модулів. Тому необхідно мінімізувати кількість внесених в існуючу розробку змін, необхідних для додавання підтримки нового протоколу.

Відзначимо, що деякі практичні завдання вирішуються за допомогою аналізу файлу зі збереженим трафіком (будемо називати такий файл мережевою трасою):

- відтворення помилок розбору;

- розробка (налагодження) розбирачів;
- розслідування інцидентів порушення інформаційної безпеки. Тому вкрай важливо забезпечити можливість використання «результатів» offline-аналізу для роботи в режимі online, тобто перенесення модулів розбору між інструментами offline- і online-аналізу.

Необхідність в проведенні розбору заголовків мережевих пакетів, як вже було зазначено, виникає при вирішенні багатьох практичних завдань. Важливо розуміти, що фахівець в області мережевої безпеки в загальному випадку може не мати високі навичками в програмуванні. Тому необхідно надати високорівневий інтерфейс (API), що дозволяє користувачеві підтримувати в рамках системи нові (зокрема закриті) мережеві протоколи.

2. ЗАСОБИ МОНІТОРИНГУ МЕРЕЖЕВОГО ТРАФІКУ

Основним інструментом для спостереження повідомлень, якими обмінюються хости є сніфер пакетів. Як випливає з назви, сніфер пакетів захоплює ("нюхає") повідомлення, передані з комп'ютера або отримані ним. Крім цього програма-сніфер, як правило зберігає та відображає вміст різних полів протоколів у цих захоплених повідомленнях. Сніфер пакетів – це пасивна програма. Вона досліджує повідомлення, що відправляються і одержуються програмами і протоколами, які працюють на комп'ютері, але ніколи не відправляє пакети сама. Отримані пакети ніколи не адресуються сніферу. Сніфер пакетів отримує копію пакетів, переданих/отриманих додатками або протоколами, які виконуються на комп'ютері.

2.1.Пакетні сніфери

Пакетний сніфер — це або програмний, або апаратний інструмент для перехоплення, реєстрації та аналізу мережевого трафіку та даних. Ці інструменти допомагають визначити, класифікувати та усунути неполадки мережевого трафіку за типом програми, джерелом та пунктом призначення. На ринку є різноманітні інструменти, більшість з яких покладаються на інтерфейси прикладних програм (API), відомі як pcap (для Unix-подібних систем) або libcap (для систем Windows) для захоплення мережевого трафіку. Тоді найкращі сніфери пакетів аналізують ці дані, що дозволяє вам точно визначити джерело проблеми та не допустити її в майбутньому. [1]

Щоб по-справжньому зрозуміти важливість сніфферів , важливо розуміти як відбувається маршрутизація в Інтернеті. Почнемо з початку. Кожен електронний лист, який ви надсилаєте, відкрита веб-сторінку та файл, яким ви ділитесь, поширюється в Інтернеті як тисячі маленьких керованих фрагментів, відомих як пакети даних. Ці пакети передаються через стек протоколів, відомий як протокол управління передачею / протокол Інтернету (TCP / IP). TCP / IP розбивається на чотири шари:

рівень протоколу додатків, рівень протоколу управління передачею (TCP), рівень інтернет-протоколу (IP) та апаратний рівень.[2]

Кожен пакет переміщується через рівень програми вашої мережі до рівня TCP, де йому присвоєний номер порту. Далі, пакет переходить на IP-рівень і отримує свою цільову IP-адресу. Як тільки пакет має номер порту та IP-адресу, він може бути відправлений через Інтернет. Надсилання здійснюється через апаратний рівень, який перетворює пакетні дані в мережеві сигнали. Коли пакет прибуває до місця призначення, дані, які використовуються для маршрутизації пакету (номер порту, IP-адреса тощо), видаляються, і пакет рухається далі через стек протоколів нової мережі. Після досягнення вершини він збирається в первісну форму .

2.1.1.Опис роботи пакетних сніферів

Пакетні сніфери працюють, перехоплюючи дані про трафік під час проходження по дротовій або бездротовій мережі та копіюючи їх у файл. Це відомо як захоплення пакетів. Хоча комп'ютери, як правило, розроблені для того, щоб ігнорувати ступінь трафікової активності від інших комп’ютерів, пакетні сніфери це перетворюють. При встановленні програмного забезпечення, мережева карта інтерфейсу (NIC) - інтерфейс між вашим комп’ютером та мережею - повинна бути встановлена в розрядний режим. Це дає змогу комп’ютеру зафіксувати та обробити через sniffer пакет все, що потрапляє в мережу.[3]

Що можна захопити, залежить від типу мережі. Для дротових мереж конфігурація мережевих комутаторів, які відповідають за централізацію зв'язку з декількох підключених пристроїв, визначає, чи може мережевий сніфер бачити трафік у всій мережі або лише на її частині. У бездротових мережах інструменти збору пакетів зазвичай можуть захоплювати лише один канал одночасно, якщо хост-комп'ютер не має безлічі бездротових інтерфейсів.

2.1.2.Переваги пакетних сніферів

Отже, у чому сенс аналізаторів пакетів? Сніффер може допомогти вам орієнтуватися на нові ресурси при розширенні пропускної спроможності мережі, керуванні пропускною здатністю, підвищенням ефективності, забезпеченням ділових послуг, підвищенням безпеки та покращенням роботи кінцевих користувачів. Що стосується великих та малих компаній, щоденні завдання можуть бути негайно зірвані проблемами ефективності, пов'язаними з мережею, додатком чи обома. Щоб відновити роботу їхньої компанії, систематики повинні мати можливість швидко визначити першопричину. Оскільки sniffers пакетів переглядають та збирають інформацію для всього трафіку по всій мережі, вони можуть оцінювати критичні шляхи мережі, щоб допомогти адміністраторам визначити, що програма чи мережа є причиною поганого досвіду користувачів. З цією інформацією, адміністратори краще оснащені для визначення та вирішення походження проблеми.

Коли користувачі повідомляють про повільність, адміністратори можуть використовувати аналіз PCAP для вимірювання часу реакції в мережі - також відомий як затримка мережевого шляху - та визначати кількість часу, необхідного для переходу пакета через мережевий шлях від відправника до одержувача. Це дозволяє адміністраторам швидко визначити причину уповільнення та виявити постраждалі програми, щоб вжити заходів.

Проаналізуйте трафік за типом. Оцінюючи проблеми з мережею та додатками, першочергове значення має трафік у вашій мережі. За допомогою правильного аналізатора IP-sniffer та пакетів трафік класифікується на типи на основі IP-адрес сервера призначення, використовуваних портів та вимірювання загального та відносного обсягу трафіку для кожного типу. Це дає вам змогу виявити надмірний рівень некомерційного трафіку (наприклад, соціальних медіа та зовнішнього веб-серфінгу). Ви також можете визначити трафік, що протікає через мережеве

посилання, а також трафік на конкретні сервери або програми для цілей управління ємністю.

Поліпшення пропускної здатності. Коли користувачі скаржаться на те, що «мережа повільна» або «Інтернет знижується», продуктивність припиняється, знижуючи рентабельність інвестицій . Щоб виправити цю помилку , вам потрібно зрозуміти, як та ким використовується пропускна здатність мережі. Сніфер пакету Wi-Fi може отримати показники продуктивності для автономних точок доступу, бездротових контролерів та клієнтів. Багато з них також пропонують моніторинг несправностей, продуктивності та доступності мережі, кореляцію даних між стековим стеком, аналіз мережевого шляху та багато іншого, щоб допомогти вам виявити потенційні проблеми та мінімізувати час простою мережі.

Поліпшення безпеки. Великий обсяг вихідного трафіку може означати, що хакер використовує ваші програми, або для спілкування зовні, або для передачі великої кількості даних.

2.2.Огляд інструментів обробки пакетів

Сьогодні на ринку є незліченна кількість інструментів, як платних, так і безкоштовних. І хоча кожен інструмент побудований на основних принципах збору мережевого трафіку, вони значно відрізняються за своєю шириною та глибиною. Багато інструментів з відкритим кодом є надзвичайно простими у своїй конструкції, і в цьому справа: ці інструменти створені для забезпечення надійного, чистого збору даних, залишаючи якнайменший слід. Якщо вам потрібні прості сніфери та швидка діагностика, безкоштовний інструмент з відкритим кодом може бути корисним. Багато - хоча і не всі - безкоштовні версії можна оновити, щоб надати додаткові аналітичні функції, якщо потрібна більша підтримка.

З такою кількістю продуктів на ринку важко дізнатися, який сніферний пакет вибрати. Незважаючи на те, що безкоштовних варіантів вистачає, покупка сніффера пакета може гарантувати, що ви озброєні

інструментом, який не тільки фіксує дані, але й пропонує інтуїтивний аналіз. Виходячи за рамки ваших базових сніферів, яких налічується десятки, ви знайдете більш надійні аналітичні інструменти для збору пакетів та мережевого збору даних. У багатьох випадках, що відрізняє ці інструменти, - це здатність проводити глибоку перевірку пакетів (DPI).

Ці великі інструменти на рівні підприємств часто оснащені для оповіщення про випадки виключення та створення інтуїтивно зрозумілих графіків та діаграм із відображенням детальних показників. Хоча вони йдуть високою цціною, вони варті своїх інвестицій.

2.2.1.TCPDUMP

Це популярний інтерфейс командного рядка (CLI) та інструмент дослідження пакетів з відкритим кодом, сумісний на платформах Unix та Linux. Він був винайдений у 1987 році в Національній лабораторії Лоуренса Берклі, а після цього опублікований через кілька років.

У ньому є бібліотека libpcap, розроблена мовою програмування на C, яка працює для збору інформації мережі. Libpcap забезпечує інтерфейс для всіх загальних платформ на базі Unix, включаючи FreeBSD та Linux. Інтерфейс libpcap в платформі Windows під назвою WinDump. Windump використовується WinPcap, який є портом Windows бібліотеки libpcap. Розробники розробляли бібліотеку libpcap як API незалежної платформи для роботи над різними програмами та усунення системної залежності для модулів збору даних у кожній програмі. TCPDump розглядається як інструмент розбору.[5]

За замовчуванням він перехоплює та друкує пакет, який захопило з мережі; інші функції, такі як зберігання, виконуються заданими командами. TCPDump працює так:

1) Читання / запис захопленого файлу з мережі в пакеті CAPture (PCAP) за допомогою команд CLI.

2) Він фільтрує пакети за деякими заданими параметрами.

3) Він друкує на екрані захоплені дані відповідно до заданих параметрів.

Це більш легкий і портативний інструмент для сніфу пакетів, оскільки він залежить лише від CLI, і мережеві адміністратори використовують його для доступу до мережевих пристроїв з віддаленого місця. На Рисунок 1 показаний трафік TCP / IP та його аналіз дослідження пакетів TCPDump, відображення адреси і вмісту трафіку даних(Рисунок 2.1.).

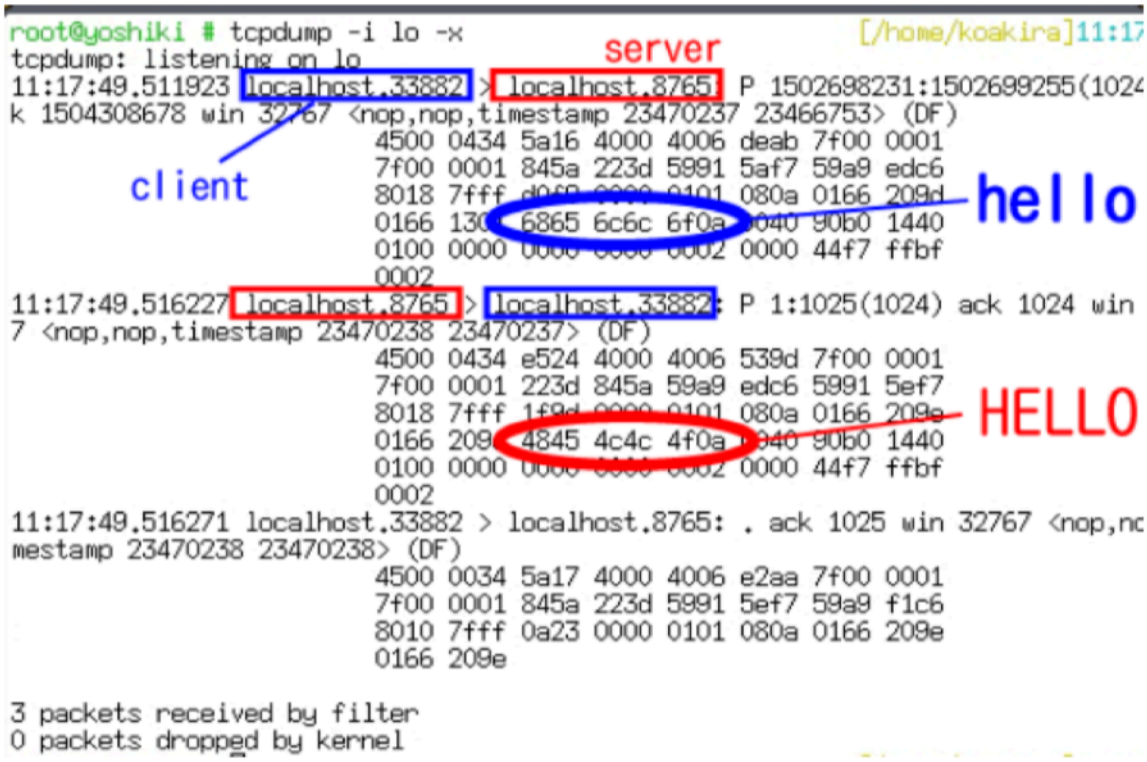


Рисунок 2.1. Огляд TCPDump, показує потік характеристик TCP/IP .

Основне обмеження TCPDump, він не надає адміністратору мережі візуального GUI захоплених даних для більшого аналізу, є тільки CLI.[9] Оскільки, це текстовий формат і користувачеві легко користуватися ним дистанційно через з'єднання Telnet. Є ще кілька недоліків у TCPDump. До них належать:

- Обмеження в аналізі трафіку, можуть застосовуватися лише протоколи на основі TCP.

- Він повідомляє лише те, що знаходить у пакетах, якщо IP-адреса підроблена в трафіку, вона не має можливості повідомляти нічого іншого .
- Пакети, заблоковані брандмауером, не відображаються.

2.2.2.Wireshark

Винайдений вченим Джеральдом Комбсом наприкінці 1997 року для перевірки та розпізнавання проблем мережі та моніторингу трафіку даних. Він назвав його Ethereal до травня 2006 року, після чого його назва змінилася на Wireshark. Це програмне забезпечення з відкритим кодом, безкоштовний інструмент і аналізатор пакетів для графічного інтерфейсу, який написаний мовою програмування на C та випущений під ліцензією GNU General Public License (GPL). Він працює на різних Unix-подібних операційних системах, включаючи Mac OS X, Linux, платформу Solaris, а також операційну систему Microsoft Windows. Інтерфейс командного рядка (CLI) Wireshark називається TShark, що дозволяє користувачеві працювати з ним за допомогою команд. Це як TCPDump з додатковим графічним інтерфейсом, підтримуючи різноманітні протоколи та можливість фільтрування та сортування. Він використовується в мережевому інструменті судового аналізу (NFAT) в організаціях.

Wireshark призначений для захоплення пакетів з працюючих мереж, а також для перегляду раніше збереженого файлу даних. Підтримуваний формат захоплення пакетів — це формат файлу "PCAP". Він відображає захоплені дані у байтовому та шістнадцятковому форматах, де відображаються різні типи використаних пакетів та протоколів. Це також дозволяє користувачеві збирати дані пакетів у потік TCP.[8]

Він має інтерфейс з трьома панелями; панель підсумків або панель списку пакетів, яка показує різні аналізовані пакети, такі як номер кадру, дата, час, адреса призначення та джерела IP, протоколи верхнього рівня, довжина пакету та інформація вмісту трафіку з кольором для кожного

захопленого типу пакету. Друга панель - це захоплені деталі пакета. Коли пакет визначається на панелі списку пакетів, деталі з'являються на наступних двох панелях; деталі та байтові або шістнадцяткові панелі. Панель деталей відображається як деревоподібна структура протоколів, захоплених для різних застосунків, таких як протокол управління передачею (TCP), протокол дейтаграм користувача (UDP), протокол повідомлення Internet Control (ICMP), протокол передачі гіпертекстового тексту (HTTP) тощо.,. Третя панель називається панеллю даних або байтом, яка показує необроблені захоплені дані, що відображають байт пакета у шістнадцятковому форматі, кодування ASCII та текстові формати.

Тут важлива примітка: для запуску інструменту Wireshark він встановлює NIC в розрядний режим, що дозволяє sniffer бачити весь трафік на цьому інтерфейсі, а не лише трафік, адресований одному з налаштованих інтерфейсів(Рисунок 2.2.). Окрім розрядного режиму, порт може показувати дзеркальне відображення до будь-яких точок мережі, коли розрядний режим охоплює її не всю.

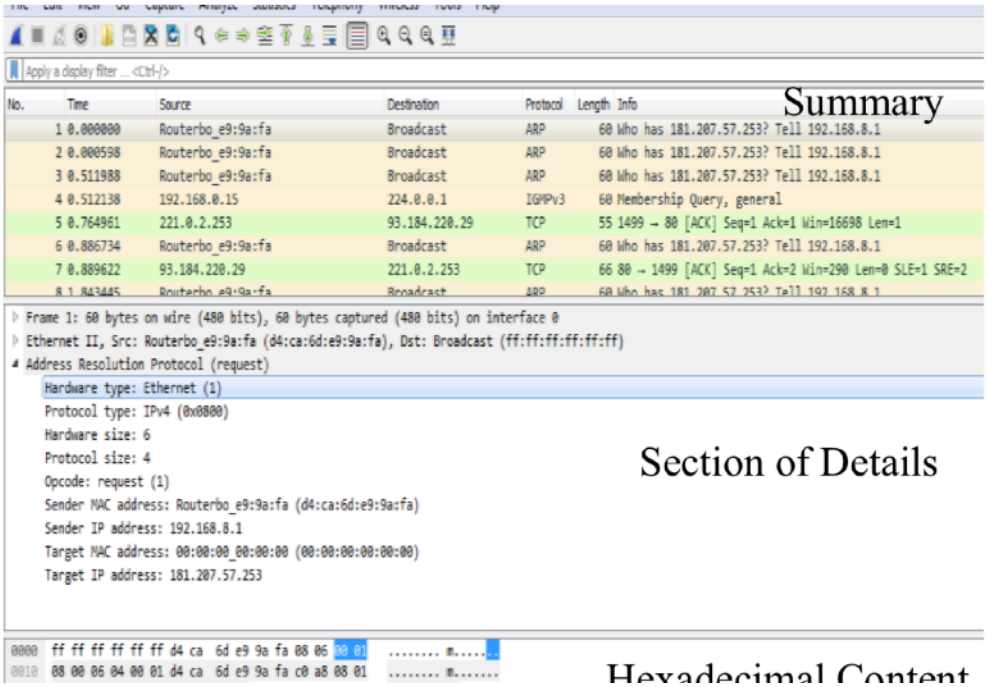


Рис 2.2. Інтерфейс інструменту для обробка пакету Wireshark.

Тут показаний інтерфейс Wireshark, що відображає три вікна; панель підсумків, деталі та байт з різними характеристиками трафіку мережі в читаному вигляді. У розділі деталей видно, що номер кадру становить 60 байт, тип мережі - це Ethernet II, тип протоколу - IPv4, а вміст корисної навантаження - ARP.[10] Розмір кадру вимірюється за допомогою модуля максимальної передачі (MTU), і цей блок визначається відповідно до використовуваного типу мережі. Наприклад, MTU для асинхронного режиму передачі (ATM) становить 53 байти , MTU в мережах Ethernet і IPv4 - 1500 байт, і в інших типах мереж використовується перемикання кадрів, які досягають 9000 байт . Отже, розмір кадру змінюється відповідно до MTU у заданому типі мережі.

Крім того, розмір кадру визначають також відповідно до використовуваної програми в мережі. У цьому випадку, це 60 байтів, в яких ARP використовується для пошуку фізичної адреси інтерфейсу маршрутизатора або хоста, коли вказана його логічна IP-адреса. В іншому прикладі дані кадру розміром 500 байт за допомогою застосованої якості обслуговування (QoS) в деяких додатках охорони здоров’я на базі UDP і зроблено висновок, що він підходить як для затримки, так і для тремтіння, а також використано деякі інші QoS що призвело до отримання кадру розміром 1500 байт. Потім кадр з'являється в інструменті дослідження пакетів відповідно до програми або протоколу, який використовується в мережі.

Обмеження Wireshark, воно потребує найкращого розуміння форматів протоколів, HTTP та Cascade Style Sheet CSS, знання мови у форматі байтів. Він використовує формат файлу PCAP для фіксації трафіку, тому він може захоплювати пакети лише тих типів мережі, які підтримують формат файлу PCAP. Ще одним недоліком є те, що Wireshark не є автоматизованим інструментом, він не є підтримкою для моніторингу тривалого часу.

2.2.3.Colasoft

Це інструмент аналізатора протоколів мережевого протоколу із закритим джерелом, призначений для роботи на платформі операційної системи Windows, що використовується для особистого використання адміністратором мережі для усунення несправностей, моніторингу та діагностики трафіку в комп'ютерній мережі. Capsa випускає безкоштовні інструменти Colasoft, що забезпечує простоту використання, аналіз пакетів у режимі реального часу та надійний криміналістичний, поглиблений аналіз протоколів, і в нього 24-годинний моніторинг мережі. Він має особливість відкриття декількох інтерфейсів в одному екземплярі, надання користувачеві графічних інтерфейсів та матричних представлень .

Він має глибокий аналіз пакетів, що показують різні характеристики з можливістю генерування звітів, журналів та попередження лише голосовими та електронними повідомленнями для ліцензованих версій. [14] Він має різноманітні функції графічного інтерфейсу, відображаючи захоплену інформацію у графіках, матриці та відповідно до кожної характеристики трафіку мережі, що показує кожен протокол, що використовується в мережі. На рисунку 2.3 показаний інструмент Colasoft та його розширені функції GUI.

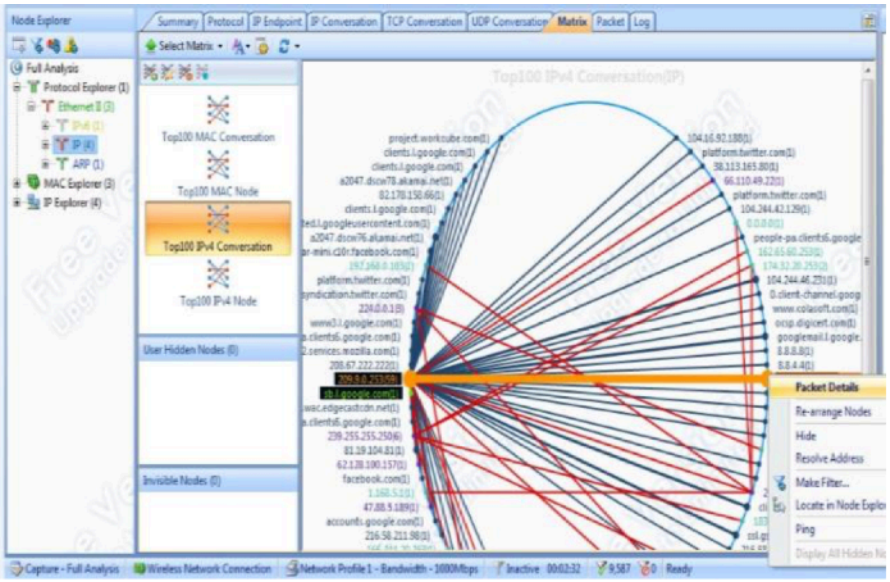


Рисунок 2.3. Інтерфейс інструменту Colasoft.

Є деякі обмеження Colasoft; це дорогий додаток, тоді як безкоштовна версія доступна, але з обмеженими можливостями, наприклад, безкоштовна версія не повідомляє користувача через електронну пошту та голосові канали. Ще два недоліки інструменту Colasoft - це те, що він працює лише на платформі операційної системи Microsoft Windows, і він підтримує лише 300 протоколів, що вважається меншим порівняно з іншими інструментами для пакетів, такими як Wireshark.

2.3.Порівняльний аналіз інструментів обробки пакетів

Для порівняння між перерахованими вище трьома методами мережевого аналізу ми залежимо від таких параметрів, як код з відкритим вихідним кодом, кількість підтримуваних протоколів, підтримувана операційна система, підтримує PCAP, користувальницький інтерфейс, вартість, форми декодування, визначені аномальні пакети, реконструюють потік TCP, тощо.

В таблиці 2.1 показано порівняння між інструментами Wireshark, TCPDump та Colasoft. З таблиці порівняння, жоден інструмент аналізу пакетів не веде його за всіма параметрами. Але порівняння з перевагами та недоліками допоможе розробникам вдосконалити інструмент дослідження пакетів, щоб подолати ці обмеження. Тут ми порівняємо інструменти, використовуючи якісні та кількісні параметри .

Colasoft має функції аналізу більш візуального пояснення зі статистикою захоплених пакетів, відображенням більшої кількості інформації про протоколи та користувацькі програми з графіками та матричним поданням для всіх підключених кінцевих точок. Додаткові функції, які Colasoft включає в себе звіти, журнали аудиту та діагностику. Усі ці функції допомагають адміністраторові мережі діагностувати проблеми мережі. Також Colasoft має потужний аналіз та інтерпретацію потоку TCP; він має універсальну пропускну здатність, мережевий трафік

та аналіз використання. [21] Він має матричне подання та функції поглибленого декодування трафіку з багаторазовою поведінкою мережевого моніторингу. Крім того, вона має візуалізацію приховування комп'ютерних мереж.

Крім того, Colasoft має більш потужну видимість та стиль Windows 7 з простими графічними екранами, інформаційними панелями та мережевим аналізатором. За допомогою цього інструменту для користувача легко і просто зробити аналіз ,просто завдання, яке ви хочете, виконується натисканням миші. Отже, Colasoft стає більш зручним інструментом для дослідження пакетів, він забезпечує легкий для читання спосіб і має декілька інтерфейсів в одному екземплярі. Графіки показують більшу візуалізацію для різних статистичних даних і властивостей мережі. Wireshark обмежений цими можливостями GUI, і він не відкриває декілька інтерфейсів в одному екземплярі.

Таблиця 2.1. Порівняльні характеристики між Wireshark, TCPDump та Colasoft Packet Sniffing Tools

Параметри	**Сніферні інструменти**		
	TCPDump	**WireShark**	**Colasoft**
Відкритий код	+	+	-
Якими операційними системами підтримується	Linux(WINDum для Windows)	Linux, Windows	Windows
Число підтримки протоколів	TCP/IP	БІльш ніж 300	300
Користувацький інтерфейс	CLI	CLI і GUI	GUI
Вартість	Безкоштовно	Безкоштовно	999 $
Libpcap основа	+	+	-
Визначення прихований даних	-	+	+
Використання місця на диску	484КБ	449МБ для Unix 89МБ для Windows	32МБ
Відображення в додатку шару протоколу	-	+	+

Параметри	Сніферні інструменти		
	TCPDump	**WireShark**	**Colasoft**
Декодування протоколу	Лише Hex, ASCII	Лише Hex, ASCII	EBDIC, Hex, ASCII
Відновлення TCP потоку	-	+(але лише форматоване)	+
Виявлення ненормальних даних	-	-(лише створює попередження)	+
Кілька інтерфейсів	-	-	+
Сповіщення знаходження	-	-	+
Відновлення HTTP веб сторінки	-	-(показує актуальний контент трафіку індивідуально)	-(показує лінки контенту трафіка індивідуально)
Мережева комунікаційна матрична мапа	-	-	+
Оцінка критичного та некритичного для бізнесу трафіку	-	+(за допомогою створення нових фільтрів та пошуку)	+(вбудована)
Можливість розробляти та налаштовувати розробниками	+(можливе налаштування користувачем під себе)	+	-(лише командою розробки capsa.co)
UDP трафік	-	+	+

У порівнянні з Wireshark, Colasoft забезпечує більшу мережеву безпеку за допомогою сповіщень про попередження через аудіо та електронні листи. Недолік Colasoft, він охоплює лише 300 протоколів, що дуже менше порівняно з Wireshark, який підтримує 1100 протоколів .

TCPDump - це дуже портативний і економічний пакет нюхає інструмент з точки зору використання пам’яті, оскільки він займає лише 484 КБ місця для встановлення. Хоча розмір інсталяційного файлу Wireshark на початку інсталяції становить 18 МБ, але після завершення інсталяції він споживає 81 Мбайт у Windows та 449 МБ дискового простору в операційній системі Linux. Місце для встановлення Colasoft становить 32 Мб. Отже, що стосується використання пам'яті, то Wireshark дуже дорогий.

Оскільки Wireshark - це відкритий код, кожен може завантажити його код і вдосконалити його. У світі існує багато універсальних розробників, які мають можливість налаштування та вдосконалення цього інструменту, в той час як Colasoft обмежений лише командою розробників компанії Capsa. Отже, Wireshark вважається хорошим інструментом дослідження пакетів для розуміння функцій програмування, і він відповідає вимогам користувачів мережі, здійснюючи налаштування без витрат. У інструменті Colasoft, якщо користувачеві потрібна певна налаштування для певної проблеми моніторингу мережі, він вимагає від компанії, яка надає оплату, за ці налаштування. За допомогою інструменту аналізу пакетів Wireshark ви можете отримати більше досвіду конфігурації TCP / IP, розуміючи структуру мережі, а також вона працює на різних платформах, включаючи Linux, Solaris, OS X і Windows.

Крім того, деякі автори проводять дослідження в цій галузі для вдосконалення інструменту Wireshark, тут вдосконалено інструмент дослідження пакетів Wireshark для виявлення вторгнення типів атак Denial of Service (DoS), особливо в тому випадку, коли можна подолати атаку затоплення ping, яка надсилає велику кількість команд ping на пристрій жертви.

2.3.1.Результати порівняльного аналізу

Усі ці інструменти мають загальні характеристики мережевих властивостей, але кожен інструмент має свою конкурентну особливість. Існують різноманітні якісні та кількісні параметри, які обговорюються та порівнюються на інструментах : Wireshark, TCPDump і Colasoft. З цих параметрів - кількість підтримуваних протоколів, відкритий вихідний код, платформа, що підтримується, бібліотека libpcap, підтримка PCAP, користувальницький інтерфейс, вартість, форми декодування, визначені аномальні пакети, мережевий зв’язок у матричній карті, реконструкція потоку TCP тощо.

Отже, кожен інструмент мережевого аналізатора не призводить до всіх мережевих параметрів. Оскільки інструмент Colasoft кращий за Wireshark у матричних та графічних звітах, Wireshark - це відкритий вихідний код, який легко розробляти та налаштовувати код всім відповідно до їх потреб, і він сумісний з різними платформами, такими як Linux та Операційні системи MS Windows, а Colasoft працює лише на операційних системах MS. З іншого боку, інструмент TCPDump - це легкий інструмент, який займає невеликий розмір простору і це конкурентна особливість, тому найкращим варіантом його дистанційного використання для моніторингу комп'ютерних мереж за допомогою інтерфейсу командного рядка. Іншим важливим фактором параметра є кількість протоколів, що підтримуються засобом сніферів пакетів. Wireshark підтримує величезну кількість протоколів понад 1000 протоколів, що є чудовим інструментом для моніторингу та контролю мереж, що використовується в різнорідних мережах, які мають різноманітні протоколи, включаючи відео та аудіо програми, в той час як інші інструменти мережевого моніторингу підтримують лише декілька таких протоколів оскільки інструмент Colasoft підтримує близько 300 протоколів, а TCPDump підтримує протокол TCP / IP і не підтримує протокол транспортного рівня протоколу користувача (UDP).

Крім того, враховуючи інші параметри, такі як вартість, Wireshark і TCPDump є безкоштовними інструментами, в той час як Colasoft є дорогим і дорожчим, ніж інші інструменти. Але Colasoft є більш сильним інструментом для виявлення ненормальних протоколів, що є конкурентною особливістю порівняно з іншими інструментами, такими як Wireshark, які лише попереджають. Інструмент Colasoft розроблений командою Capsa, який став хорошим графічним інтерфейсом і має більше функцій безпеки. Інтерфейс фільтрації також розглядається як конкурентна особливість в інструменті Colasoft, який є з графічним інтерфейсом та зручним для користувача, що дозволяє користувачеві легко фільтрувати та аналізувати

протоколи та трафік даних. В таблиці 2.2 показано найкраще використання інструменту дослідження пакетів для кожного ресурсу мережі.

Таблиця 2.2. - Найкраще використання інструментів обробки пакетів; Інструменти Colasoft, Wireshark та TCPDump.

Мережеві параметри	Сніферні інстументи
Захист комп'ютерних мереж	Colasoft
GUI	
Виявлення нетипових пакетів	
Мережеві сповіщення	
Розмір пакетів	
Мережеве спілкування	
Кілька або один інтерфейс	
Декодування протоколів Hex, ASCII, EBDIC	
Виявлення ненормальних пакетів	
Визначення пакетів з підробленими даними	
Відображення шарів протоколів додатку у OSI 7 моделі	
ПІдтримка OS	Wireshark
Налаштування і розробка всіма розробниками	
Час відповіді	
Пропускна здатність	
Швидкість обробки	
Число протоколів, що підтримуються	
Портативний і простий контроль віддаленого доступу	TCPDump

Найкраще використовувати Colasoft для сигналів про ненормальні та підроблені пакети, це забезпечує більшу безпеку та інтерфейси GUI. Хоча Wireshark підходить для навчання програмістами та розробниками, завантажуючи вихідний код і налаштовуйте його відповідно до потреб мережевого моніторингу. Інструмент TCPDump більше підходить для

віддаленого логічного контролю доступу для моніторингу мережі за допомогою CLI.

Крім того, деякі інші автори перевірили деякі мережеві параметри для порівняння між засобами дослідження пакетів; результат показаний у наступних пунктах [22].

- Час відповіді

Він визначається як тривалість періодів часу (вимірюється у одиницях часу) для певної конкретної події. Автори роблять висновок, що час реакції Wireshark менше часу реакції Colasoft.

- Пакети за секунди (PPS)

Це кількість переданих пакетів за одну секунду. Добре видно, що Wireshark має менші втрати пакетів, ніж Colasoft. Отже, Wireshark вважається кращим порівняно з Colasoft для повторної передачі пакетів.

- Розподіл розміру пакету

Менший розмір пакетів може призвести до меншої напруги в мережі, тоді як великий розмір пакетів збільшує навантаження на мережу. Після експерименту вони дійшли висновку, що розмір пакету довжини у Wireshark становить 558,76 байт, тоді як у Colasoft - 434 . Отже, Colasoft надсилає довжину пакету середнього розміру; це краще, ніж інструмент Wireshark для завантаження комп'ютерної мережі. Colasoft Capsa не підкреслює систему та мережу.

- Пропускна здатність (біт на секунду)

Це обсяг даних, оброблений системою, виміряний у секунду. Після експерименту показано, що пропускна здатність у Colasoft Capsa є великим діапазоном і він швидко змінюється. Ці випадкові зміни погано впливають на систему мережі, оскільки вони заважають роботі системи та комп'ютерної мережі. Тоді як Wireshark має системність та хороший діапазон моделей мережі комп'ютера. Середній біт на секунду (bps) у

Wireshark становить 115,398 кбіт / с, а в Colasoft - 6,34 кбіт / с. Отже, Wireshark має більшу пропускну здатність більше, ніж Colasoft Capsa, тоді Wireshark має більшу продуктивність з постійними коливаннями, а також, не бачачи високих відсічок Bps.

2.4.FATT(Fingerprint all the things)

2.4.1.Опис FATT

На основі попереднього аналізу існуючих рішень з точки зору функціоналу пакетного сніфера було створено власний — FATT(Fingerprint all the things). Розроблено скрипт для вилучення метаданих та відбитків(програмний код див. додаток Б). Загальний опис створеного продукту:

- Розроблене рішення є кросплатформним(Linux, macOS та Windows).
- З можливістю вилучення метаданих та цифрових відбитків з різних джерел трафіку з поточного та файлу пакетних даних мережі(.pcap).
- Реалізована підтримка протоколів SSL/TLS, SSH, RDP, HTTP. Для вилучення цифрових відбитків пристроїв було обрано такі методи реалізації для відповідних протоколів:

1. JA3 для протоколу TLS.

   JA3 — це метод створення відбитків SSL/TLS, який повинен бути легким для інтеграції на будь-якій платформі і може бути легко застосований для дослідження загроз. [15,16] Це набагато ефективніший спосіб виявлення зловмисної активності через SSL, ніж індикатори компрометації на основі IP або домену. Оскільки JA3 виявляє клієнтську програму, не має значення, чи зловмисне програмне забезпечення використовує DGA (алгоритми генерації домену) або різні IP-адреси для кожного хоста, JA3 може виявити саме шкідливе програмне забезпечення

на основі того, як проходить взаємодія, а не на тому, які компоненти взаємодіють. JA3 також є чудовим механізмом виявлення в замкнених середовищах, де дозволено встановлювати лише кілька конкретних програм. У таких типах середовищ можна скласти білий список очікуваних програм, а потім попередити про будь-які інші звернення JA3.

2. HASSH для протоколу SSH

"HASSH" - це мережевий стандарт відбитків, який може бути використаний для ідентифікації конкретних реалізацій клієнта та сервера SSH. Відбитки можна легко зберігати, шукати та ділитися ними у вигляді відбитка MD5. "hassh" і "hasshServer" - хеші MD5, побудовані з певного набору алгоритмів, які підтримуються різними SSH-клієнтськими та серверними програмами. Ці алгоритми обмінюються після початкового тристороннього handshake-у TCP як пакети з чітким текстом, відомі як повідомлення "SSH_MSG_KEXINIT", і є невід'ємною частиною налаштування остаточного зашифрованого каналу SSH. Існування та впорядкування цих алгоритмів є досить унікальним, щоб його можна було використовувати як відбиток, щоб допомогти визначити базовий додаток для клієнтів і серверів або унікальну реалізацію, незалежно від нібито ідентифікаторів вищого рівня, таких як рядки "Клієнт" або «Сервер»(Рисунок 2.4).

3. RDFP для протоколу RDP.

RDFP - експерементально розроблений RDP відбиток для стандартного RDP протоколу(для інших варіаій безпеки RDP-протоколу використовується TLS, таким чином відбтиток може бути захаплений за допомогою JA3).

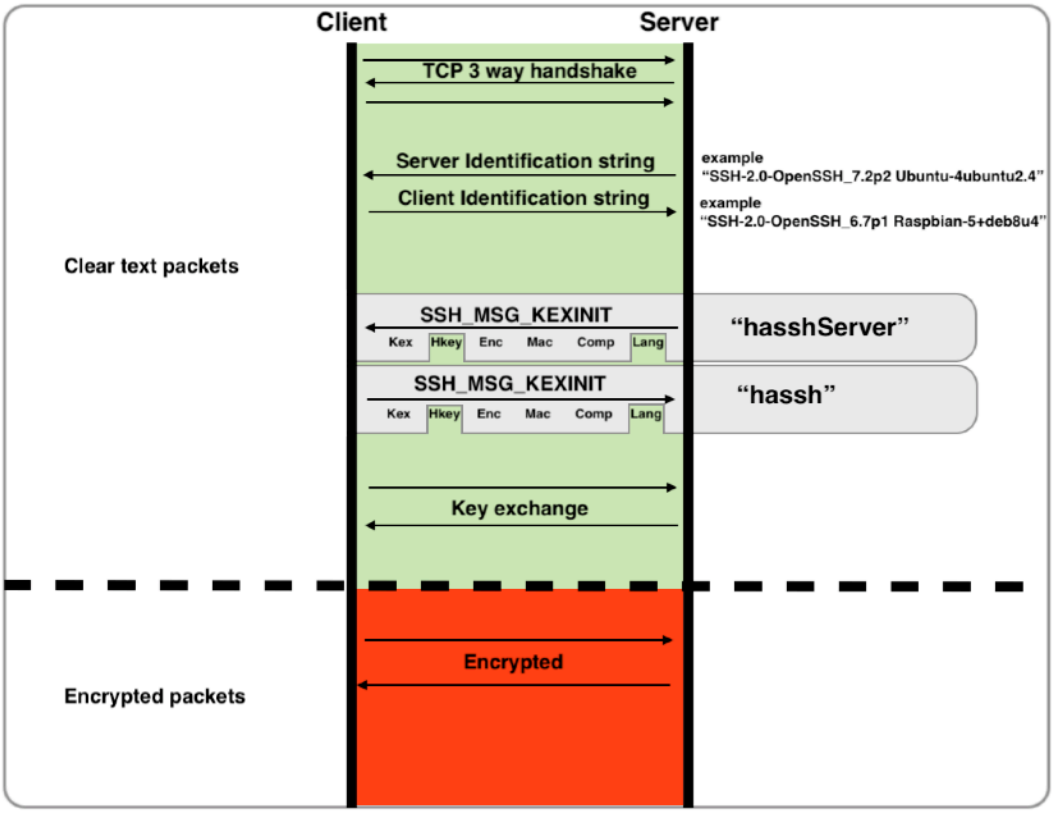


Рисунок 2.4. Принип роботи HASSH

2.4.2.Використання FATT

Основним варіантом використання даного ПЗ це моніторинг приманок (honeypots), також продукт можна використовувати для таких цілей, які перед нами ставить мережевий криміналістичний аналіз.

Як користуватися fatt:

usage: fatt.py [-h] [-r READ_FILE]
[-d READ_DIRECTORY] [-i INTERFACE]
[-fp [{tls,ssh,rdp,http,gquic}[{tls,ssh,rdp,http,gquic} ...]]]
[-da DECODE_AS]
[-f BPF_FILTER]
[-j][-o OUTPUT_FILE]
[-w WRITE_PCAP] [-p]

Необов'язкові аргументи:

- -r, --read_file — ім’я файлу пакетних даних мережі(.pcap)
- -d, --read_directory — директорія файлу пакетних даних мережі(.pcap);
- -i, --interface — інтерфейс, який досліджується;
- -fp, --fingerprint — види протоколів, які включені(за замовчуванням - всі);
- -da, --decode_as — словник типу {decode_criterion_string: decode_as_protocol}, який використовується для того, щоб повідомити tshark як розшифрувати протоколи в незвичайних ситуаціях;
- -f, --bpf_filter — BPF фильтр(лише для режиму поточного трафіку);
- -j, --json_logging — формат виведення JSON;
- -o, --output_file — файл журналу(за замовчуванням fatt.log);
- -w --write_pcap — запис захоплених пакетів з поточного трафіку до файлу пакетних даних мережі;
- -p, --print_output — виведення вихідних данних.

Результат роботи програми при аналізі поточного трафіку (див. Додаток В). Відповідний розширений JSON файл (див. Додаток Г).

Результати роботи FATT співпали з відповідними у Wireshark - з цього можна зробити висновок про правильність реалізації підтримки відповідних протоколів.

Розглянемо приклади роботи для файлів пакетних даних мережі(.pcap):

- для вразливості CVE-2020-0708 RDP (BlueKeep) (див. Додаток Д).
- перевірка за допомогою іншої вразливості CVE-2020-0708 PoC (див. Додаток Е). Цього разу ми не бачимо повідомлення RDP ClientInfo, оскільки PoC використовує TLS (не стандартний протокол безпеки RDP). Таким чином, ми можемо просто побачити повідомлення із запитом на підключення, але якщо ви декодуєте пакет як TLS, ви можете бачити відбитки клієнта TLSHello та JA3. Можна декодувати певний порт як інший протокол(див. Додаток Є).

2.4.3.Вектор розвитку FATT

Вектор розвитку створеного программного продукту:

- Підтримка нових протоколів (ETF QUIC, MySQL, MSSQL, SMTP, SMB).
- Реалізація модуля візуалізації(UI частини програмного продукту).

2.5.Висновки

Розглянувши та порівнявши поведінку вже існуючого програмного забезпечення для аналізаторів мережевого трафіку, таких як Wireshark(раніше відомий як ethereal), TCPDUMP та Colasoft. Кожна з цих програм пропонує різні функції та обмеження, відповідно наведеної таблиці 3.3.2 для використання відповідного програмного продукта в залежності від потреб.

Аналізуючи існуючі системи, розробили відповідний аналог з базовим функціоналом, але з унікальними методами створення відбитків для протоколів, ми зробили наступні висновки:

- Сніфери видають лише журнал даних, який повинен аналізувати мережевий адміністратор, щоб знайти помилку або атаку на мережевий адаптер.
- Поточні системи здатні показувати лише журнали пакетів.
- Обмеження аналізу на основі протоколу включають той факт, що це надзвичайно трудомістко захопити кожен пакет, вивчити їх, розібрати кожен та вручну здійснити дію на основі інтерпретацій аналізу.

3. МОДЕЛЮВАННЯ МЕРЕЖЕВОГО ТРАФІКУ

При аналізі мережевого трафіку можна можна діагностувати проблеми та виявляти вразливості працюючої системи. Після внесення відповідних змін, які закривають ці вразливості та вирішують проблеми. Постає питання відповідної перевірки правильності та повноти рішення, маючи відповідний дамп після аналізу, ми можемо це зробити. Але як уникнути відповідних проблем на етапі розробки? В цьому випадку нам на допомогу приходить техніка моделювання трафіку.

3.1.Огляд генераторів мережевого трафіку

Існуючі генератори трафіку зосереджуються насамперед на підтримці статистичного розподілу різних характеристик мережевого трафіку, де можна розпізнати три основні підходи. Генератори трафіку на рівні пакетів, такі як iPerf [5], базуються на часі взаємодії і розмірі пакету. Розмір кожного відправленого пакету, а також час, що минає між наступними пакетами, вибираються користувачем, як правило, шляхом визначення статистичних розподілів для кожної змінної. Ці генератори критикуються за неточність [9]. Крім того, вони використовуються для перевірки працездатності та масштабованості мережевих інструментів, а трафіку, який вони генерують, бракує багатства та різноманітності пакетних потоків, що спостерігаються в реальних організаційних мережах; такі інструменти зосереджуються на тестах ефективності, а не на відтворенні характеру трафіку.

Генератори трафіку на рівні додатків імітують поведінку конкретних мережевих додатків щодо трафіку, який вони генерують. Наприклад, Surge [6] - це генератор завантаженості мережі, побудований за допомогою інтеграції розподілів функцій, витягнутих із використання цієї мережі. Ці інструменти, як правило, фокусуються на створенні послідовностей запитів на рівні додатків, які призводять до мережевого трафіку з аналогічними статистичними даними трафіку, ніж реального трафіку в

модельованому додатку. Хоча корисні для оцінки поведінки та продуктивності хост-систем, ці інструменти відтворюють лише один тип трафіку додатків, а не різноманітність трафіку, що переглядається в Інтернеті.

Генератори трафіку на рівні потоку, такі як Harpoon [7], відтворюють потоки трафіку (де потік визначається як серія пакетів між заданою парою IP/порт), що є представником тих, що переглядаються в Інтернет-трафіку, виходячи з восьми розподільних характеристик TCP та UDP потоки. Параметри цих дистрибутивів можуть бути автоматично вилучені з даних, зібраних з живого трафіку. Ці функції дозволяють Harpoon створювати статистичні навантаження, які не залежать від будь-якої конкретної програми. Тим не менш, статистика лише для восьми ресурсів зберігається в генерованому трафіку. Такі рамки, як Swing [8], намагаються подолати цю слабкість, враховуючи взаємодію між декількома шарами (користувачами, сесіями, з'єднаннями та мережевими характеристиками), щоб краще зберегти більше властивостей генерованого трафіку. Тим не менш, всі вищезазначені методи в основному зберігають статистику різних характеристик руху, але не мають можливості зберегти загальні послідовності в межах трафіку.[18]

Дутта та ін. [4] представити основу для моделювання ботів користувачів, наслідуючи дії реальних користувачів, і продемонструвати його успішну реалізацію для виявлення вторгнень. Хоча їх робота базується на заздалегідь визначених правилах, ми підійдемо до аналогічного завдання з машинними методами навчання, які дозволять автоматизувати витяг реальної поведінки користувачів.

3.2.Формування ідеї

Використання генераторів мережевого трафіку в останні роки зростає, головним чином, через обмеження конфіденційності персональної інформації, які обмежують використання реальних даних. Синтетичні дані

можна вважати «фальшивими» даними, створеними з «реальних» даних. Перевага його випливає з його основи в реальних даних і реальних розподілах, які роблять його практично невідмінним від вихідних даних. Її імпульс, в умовах конфіденційності, випливає з того, що багато разів юридично неможливо ділитися реальними даними, але насправді анонімізовані дані є недостатньо корисними. У ці моменти встановлення синтетичного набору даних може представляти найкраще рішення обох світів - дані, якими можна ділитися, але все ж нагадують вихідні дані.

«Корисність» синтетичних даних була підтверджена такими дослідженнями, як [1–4]. У цьому дослідженні ми вивчаємо використання синтетичних даних для іншого типу даних, даних мережевого трафіку. Основна ідея запропонованого методу полягає в підготовці генеративної моделі, заснованої на даних обмеженої кількості користувачів, що погоджуються, а потім відповідно генерувати більш широкий і різноманітний трафік.

Створений трафік, який зберігає важливі особливості початкового трафіку, включаючи розподіл різних користувачів, додатків та мережевих властивостей, може бути корисним у багатьох напрямках. Наприклад, генерований трафік може слугувати вхідним фактором для аналізу вищого рівня, класифікації потоку та виявлення аномалії. Генератори мережевого трафіку також критично важливі в умовах тестування, де їх можна використовувати для проведення аналізу чи для оцінки поведінки та продуктивності нових систем та реального мережевого обладнання в корпоративних середовищах. Наприклад, інструмент вимірювання пропускної здатності може використовуватися для генерованих даних, що підтримують кількість пакетів. Ще одне використання генерованого трафіку в симульованих середовищах, де потрібен реалістичний фоновий трафік. Без можливості постійно створювати нові тестові умови (генеровані дані) в мережі, системи ризикують не впоратися з несподіваною поведінкою і погано працювати. Тому потрібні відповідні

методи для створення масштабованого, регульованого та репрезентативного мережевого трафіку.

Існуючі рішення [5–8] були зосереджені на генеруванні статистично репрезентативного трафіку, але, наскільки нам відомо, не існує системи, яка б також зберігала послідовність моделей мережевої діяльності. Щоб краще зрозуміти значення збереження послідовностей мережевої активності, розглянемо наступний приклад: завантаження PDF-файлу з певного сайту зазвичай виконується після читання електронної пошти з папки "Вхідні" Gmail. Навіть якщо аналіз записаного трафіку покаже, що ця закономірність є статистично значущою, жоден із існуючих методів генерування штучного трафіку не збереже її. Тому система виявлення вторгнень (IDS), оцінена за допомогою генерованого трафіку, може не виявити можливе джерело атаки та виявити слабкішу ефективність, ніж це було б для вихідного трафіку.

Це призводить до основної мети дослідження в цьому розділі. Пропонується ідея Генератора мережевого трафіку, основу для збурення записаного мережевого трафіку з метою генерування різноманітного, але реалістичного фонового трафіку для сценаріїв моделювання мережі. NTG використовуватиме сотні функцій для підтримки якомога більшої кількості атрибутів вихідного трафіку, що забезпечить надійне рішення, яке не обмежується конкретним протоколом або додатком. Пакетні потоки кластеризовані для ідентифікації подібних мережевих дій, а потім методи машинного навчання використовуються для створення послідовних зразків мережевих дій, які будуть утримуватися в генерованому трафіку.

3.3.Прототип генератора мережевого трафіку

Прототип генератора мережевого трафіку включає чотири фази, що складаються з взаємопов'язаних наборів компонентів, як зображено на

рисунку 3.1., попередня обробка мережевого трафіку, кластеризація потоків для подібних дій, моделювання послідовностей діяльності та генерація трафіку.

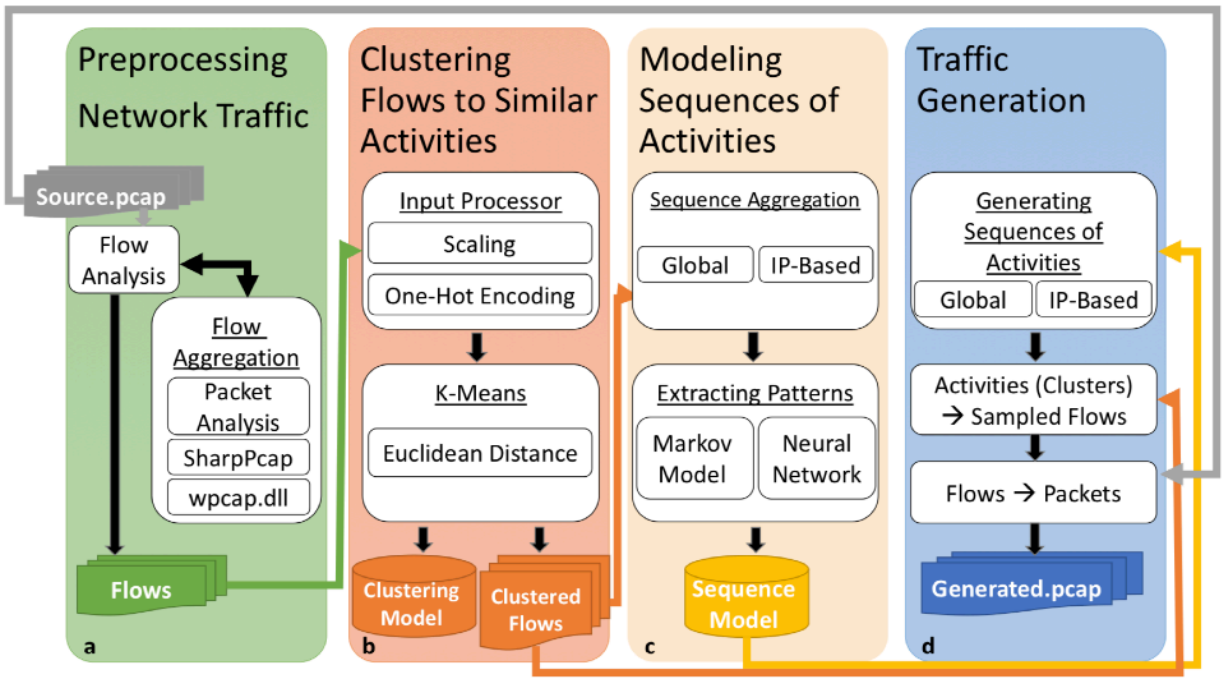


Рисунок 3.1. Фази генератора мережевого трафіку

- вихідний трафік попередньо обробляється на агреговані потоки;
- потоки кластеризовані в групи мережевої діяльності;
- модель послідовності навчається для витягування загальних зразків на основі послідовностей мережевих дій (для цього завдання підтримується кілька методів);
- генеруються нові послідовності мережевих дій і потім перетворюються на вибіркові потоки пакетів, які відповідають кожній діяльності.

На етапі попередньої обробки ми зосередилися на двох найпоширеніших протоколах прикладного рівня (DNS та HTTP). Тим не менш, прототип легко розширити, щоб включити додаткові протоколи шляхом вилучення додаткових функцій. Після попередньої обробки записи потоків аналізуються далі для вилучення подібних дій. На наступній фазі послідовності діяльності моделюються за допомогою моделі Маркова або

нейронних мереж, і, нарешті, на останній фазі треновану модель послідовностей використовують для генерування штучного трафіку.

3.3.1.Попередня обробка мережевого трафіку

Основна сутність, що розглядається — потік трафіку; цей термін відноситься до послідовності пакетів, які були передані між двома вузлами протягом одного сеансу, при цьому два вузли являють собою унікальний 4-пакет, що складається з вихідних та цільових IP-адрес та портів. Сеанс TCP починається з успішного handshake-у і закінчується таймаутом або пакетом із прапорцем RST або FIN від однорангового. Оскільки в цьому протоколі вузли посилають пакети по черзі, генеруються два потоки, по одному для кожного напрямку сеансу. Сеанс UDP складається з усіх пакетів, що надсилаються від клієнта до сервера до тих пір, поки не буде досягнуто визначеного часу простого зв'язку або максимальної граничної тривалості. Тому для кожного сеансу генерується один потік

Визначили 205 унікальних функцій, які можуть формувати поведінку мережевого трафіку._Ми витягли функції з трьох рівнів:

1) рівень пакету;

2) рівень потоку ;

3) рівень застосування.

Виділені функції детально описані в таблиці 1. Особливості рівня пакету витягуються з кожного пакету на рівні транспортного шару. Ми вилучили п'ятнадцять атрибутів на основі TCP та одну функцію на основі UDP. Кожна необроблена функція була перетворена в набір агрегативних ознак, які описують кожну особливість у пакетах заданого потоку. Наприклад, "розмір пакету" дістається до одинадцяти сукупних ознак, таких як середній розмір пакету (середня кількість розміру пакету в потоці), ентропія розміру пакета і т.д. Особливості рівня потоку, природно, витягуються з потоку. Ми витягли шість функцій рівня потоку; прикладом такої функції є кількість пакетів у потоці. Особливості рівня додатків - це

функції, які відображаються на рівні додатків, вони стосуються лише певних програм. Наприклад, агент HTTP користувача буде вилучений для потоків, які підходять до HTTP-додатків. Ми витягли вісім функцій на основі DNS, вісім функцій на основі HTTP та сім функцій на основі SSL

У таблиці 3.1 наведено кілька прикладів функцій, які відображаються в різних протоколах мережевого трафіку.

Таблиця 3.1 — Функцції на різних рівнях протоколів

Level	Features
Flow	time of day, packet interarrival time, the number of packets
TCP	time to live, seq num, ack num
UDP	checksum invalid
DNS	additional records, canon names, response count
HTTP	cookie, unique content types, bytes

3.3.2.Кластеризація потоків

Оскільки потік представляє певну мережеву діяльність від початку до кінця, ми можемо групувати подібні дії в кластери, які представляють типи мережевих дій. Цей крок необхідний для того, щоб дозволити виготовлення шаблонів послідовностей мережевих дій. Доступні різноманітні алгоритми кластеризації. Вибрано алгоритм k-means, оскільки це швидкий і простий алгоритм кластеризації. Для вимірювання відстані між потоками необхідно вибрати міру відстані. Ми приймаємо традиційну евклідову відстань, яка є однією з найбільш часто використовуваних метрик для кластерних проблем. Враховуючи набір потоків F = {f1, .., fn}, fi ∈ RM, з характеристиками M на один потік, відстань між двома потоками становить:

$$dist(f_i, f_j) = \sqrt{\sum_{d=1}^{|M|} (f_i^d - f_j^d)^2} \tag{3.1}$$

Алгоритм k-means ділить набір потоків на K непересічних кластерів. Для кожного кластера алгоритм максимізує однорідність усередині кластера, мінімізуючи суму квадратичних помилок відстані між кожним потоком fj та центром його кластера ci:

$$E = \sum_{i=1}^{K} \sum_{f_i \in c_i} (dist(c_i, f_j))^2 \tag{3.2}$$

Алгоритм k-means починається з випадкової ініціалізації центрів кластерів і спрямовується до мінімальної похибки, де в кожній ітерації потоки присвоюються кластерам з найближчими центрами, після чого центри кластерів перераховуються. Зазвичай алгоритм конвергується в стабільні кластери в межах невеликої кількості ітерацій. Ми провели кілька додаткових кроків попередньої обробки, щоб переконатися, що набір потоків є відповідним завданням кластеризації; це включає перетворення номінальних ознак у числові ознаки методом одноразової бінаризації, а також масштабування ознак до одиничної дисперсії.

3.3.3.Моделювання послідовностей мережевої діяльності

Потрібно розглянути різні варіанти агрегації моделей послідовності мереж. У зв’язку з цим ми зіткнулися з деякими питаннями. Чи має сенс підтримувати послідовність дій, що виконуються всіма членами мережі, чи слід зберігати послідовності дій, виконані парами комунікаційних IP-адрес (наприклад, послідовність дій від вихідного IP-адреси до певного сервера призначення)? Ми вивчаємо обидва ці параметри агрегації моделі послідовності мережевих дій. Перший (глобальний) відноситься до всього трафіку як до однієї послідовності, а другий (на основі IP) стосується кожної пари комунікаційних IP-адрес як окремої послідовності. У цьому

дослідженні ми, серед іншого, вивчаємо, яка з двох агрегацій більше підходить для наших цілей.

Послідовність s - це упорядкований перелік мережевих дій, представлений ідентифікаторами кластерів, підключених до потоків, що відбувалися послідовно: s = {c1 → c2 →. . . → ci}, де i ∈ K. У глобальній агрегації всі потоки трафіку належать одній послідовності, тоді як в агрегації на основі IP потоки, що належать одній і тій же послідовності, поділяють джерело та цільові IP адреси. Час першого пакету в кожному потоці ti диктує порядок дій у послідовності. Існує перехід між двома послідовними діями в межах однієї послідовності; час переходу між i -ою активністю і наступною активністю дорівнює ti + 1 - ti. Послідовні стани можуть бути однаковими (наприклад: c3 → c2 → c2).

Рисунок ілюструє різницю між агрегацією послідовностей на основі глобальної та IP-адреси. У глобальній агрегації всі потоки складаються з однієї послідовності (жовта стрілка вказує їх порядок). У агрегації на основі IP створюється послідовність для кожної унікальної пари вихідних та цільових IP-адрес(Рисунок 3.2.). Оскільки в наборі прикладів є дві унікальні пари, утворюються дві послідовності: перша складається з трьох потоків, витягнутих з трафіку, які пройшли з 1.1.1.2 до 2.2.2.1 (позначені червоним кольором), а друга складається з двох потоків, які пройшли від 2.2.2.3 до 2.2.2.1 (позначено синім кольором). У агрегації на основі IP ми додаємо стартовий і кінцевий стани навколо кожної послідовності, щоб дати змогу вибирати початковий стан і передбачати кінець послідовності (вивчення розмірів послідовностей).[27]

Шукаючи сильний алгоритм вилучення моделі послідовностей, у цьому дослідженні ми вивчаємо два методи: модель Маркова(див. Додаток Ж) та модель нейронної мови(див. Додаток З).

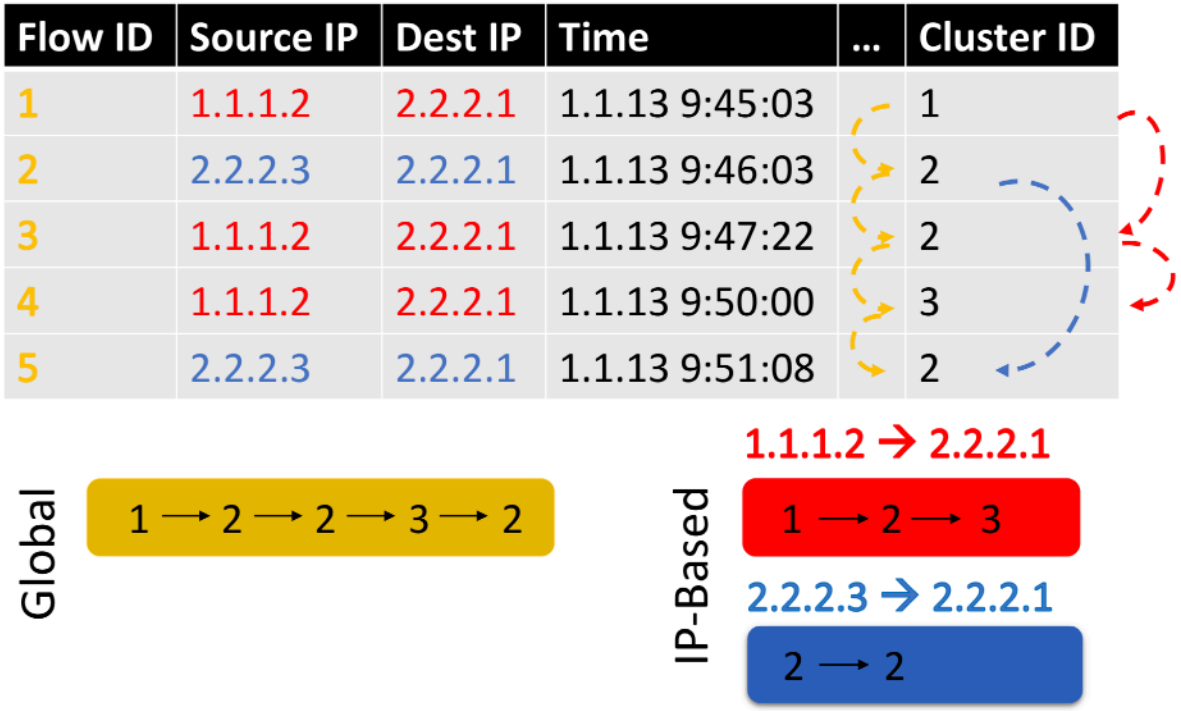


Рисунок 3.2. Дві послідовності агрегації: на основі глобальної та IP-адреси. У глобальній агрегації (жовтий) всі потоки в трафіку належать одній і тій же послідовності, тоді як в агрегації на основі IP (червоний і синій) створюється послідовність для кожної унікальної пари IP-адрес джерела та призначення. Послідовність складається з ідентифікаторів кластера, приєднаних до відповідних потоків і впорядкована часом першого пакету в кожному потоці.

3.3.4.Генерація мережевого трафіку

Ми пропонуємо генеративний метод, відповідний навченим моделям. По суті, нова послідовність мережевих дій (представлена ідентифікаторами кластера) створюється шляхом "склеювання" дуже коротких та фрагментів мережевих дій, які перекриваються, що часто часто зустрічаються у навчальних даних.[24] Правила отримання ймовірності наступної мережевої активності залежать від використовуваного методу (модель Маркова або модель нейронної мови) і є дещо неявними.

З метою генерації мережевого трафіку створюються послідовності дій (кластери); потім потоки, відповідні кожному кластеру, відбираються з вихідного трафіку, і в кінцевому підсумку пакети цих потоків передаються

в порядку, зазначеному генерованими послідовностями. У глобальній агрегації послідовностей модель послідовності навчається на потоках всіх членів мережі, і відповідно генерується єдина глобальна послідовність, яка імітує мережеві дії всіх однорангових мереж, упорядкованих за часом їх виникнення. Послідовність завершується, коли досягнутий приблизний час закінчення; отже, на розмір послідовності в основному впливають розрахункові часи передачі, які базуються на статистиці часу переходу, яка була вилучена з вихідного трафіку. У агрегації на основі IP створюється послідовність для кожної пари IP-адрес (адреси регенеруються), де послідовності закінчуються, коли доходить до стану кінця. Використовуючи модель нейронної мови, контекст m попередніх кластерів повинен бути оновлений у кожній ітерації для прогнозування наступного кластеру; при використанні моделі Маркова попередній кластер оновлюється таким же чином. Алгоритм генерації трафіку викладений в формальному описі нижче (рисунок 3.3)

```
1  D* ← ExtractFlows(D);
2  D* ← AttachClusters(D*, M_Clusters);
3  t_start, t_end ← FirstTime(D), LastTime(D);
4  trans ← ExtractHist(TransitionTimes(D*), bins = √|D*|); // transition times between flows
5  if using global aggregation then
6      c ← SampleFirstCluster(M_Sequences);
7      t ← t_start;
8      while t < t_end do
9          f ← SampleFlow(D*,c);
10         t ← t+ Sample(trans);
11         context ← Update (context,c);
12         c ← PredictCluster(M_Sequences,context);
13         Add GetPackets(f,D) to results set;
14 else
       // If using IP-based aggregation
15     for each pair IP_src,IP_dest in D do
16         context ← Update (context,start);
17         c ← PredictCluster(M_Sequences,context);
18         IP*_src,IP*_dest ← GenerateIPs();
19         t ← t_start;
20         while c! = end do
21             f ← SampleFlow(D*,c);
22             t ← t+ Sample(trans);
23             Add SetIPs(GetPackets(f,D),IP*_src,IP*_dest) to results set;
24             context ← Update (context,c);
25             c ← PredictCluster(M_Sequences,context);
```

Рисунок 3.3. Алгоритм генерації мережевого трафіку

Його вхід містить дані мережевого трафіку (D), які використовуються для реконструкції записаних пакетів при генерації нового трафіку, а також модель кластеризації (MClusters) та модель послідовностей (MSequences).

3.4.Методи оцінки генератора трафіку

Нижче буде використано оцінку запропонованої нами основи та порівняння досліджених методів. Ми використовуємо бал Silhouette, щоб оцінити, наскільки добре ми кластеризували потоки.

Багато функцій було витягнуто з мережевого трафіку, і щоб продемонструвати, наскільки добре збережений метод кожного з них, ми використовуємо тести гіпотез, щоб перевірити, чи джерело та синтезовані дані кожної функції походять з одного розподілу.[25] Ми також використовували n-гранні невдачі, щоб перевірити, чи збережено розподіл послідовності значень ознак.

3.4.1.Оцінка силуету

Оцінка силуету вимірює, наскільки член схожий на його кластер (згрупованість) порівняно з іншими кластерами (розділення). Оцінка коливається від −1 до +1, при цьому більш високі бали вказують на те, що член добре згрупований (подібно до самого кластера та далеко від сусідніх кластерів). Високі бали для більшості членів кластеру вказують на те, що конфігурація кластеру є відповідною.

Оцінка силуету для одного потоку fi розраховується так:

$$s(i) = \frac{b(f_i) - a(f_i)}{max(a(f_i), b(f_i))} \tag{3.3}$$

де a (fi) - середня відстань від fi до всіх інших потоків всередині його кластеру, a b (fi) - найменша середня відстань fi до всіх потоків у будь-якому іншому кластері (відстань до другого найближчого кластера). Ми використовували середній бал силуету кластерних потоків для оцінки структури конфігурації кластеру та налаштування кількості кластерів.

3.4.2.Двопробний тест Колмогорова-Смірнова

Тест гіпотези К.С. (Колмогорова-Смірнова) — непараметричний тест рівності безперервних одновимірних розподілів ймовірностей, який можна використовувати для порівняння двох вибірок. Він кількісно визначає відстань між емпіричними функціями розподілу двох зразків. Нульовий розподіл цієї статистики обчислюється під нульовою гіпотезою про те, що вибірки беруться з одного розподілу. Ми використовуємо цей тест, щоб перевірити, чи неперервна функція має однаковий розподіл як у вихідному, так і у генерованому трафіку.

3.4.3.Випробування K-mean Андерсона-Дарлінга

Тести гіпотез AD(Anderson-Darling) використовуються для визначення того, чи можна оцінити кілька колекцій спостережень як таких, що надходять із однієї сукупності, де функцію розподілу не потрібно вказувати. Це модифікація тесту KS. Ми використовуємо цей тест, щоб перевірити, чи має номінальна ознака однаковий розподіл як у вихідному, так і у генерованому трафіку.

3.4.4.Заплутаність

Заплутаність вимірює успіх розподілу ймовірностей або моделі ймовірності передбачення вибірки. Він може бути використаний для порівняння ймовірнісних моделей. Низька заплутаність вказує на те, що розподіл ймовірностей добре прогнозує вибірку. При обробці природною мовою заплутаність часто використовується для оцінки мовних моделей, порівнюючи n-гранні невдачі в оригінальних та створених даних. Цю ідею

адаптуємо для вивчення збереження послідовностей мережевих дій для кожного IP-джерела.

Заплутаність дискретної випадкової величини X визначається як:

$$2^{H(X)} = 2^{-\sum_x p(x) log_2(p(x))} \tag{3.4}$$

де H (p) - ентропія (у бітах) розподілу по кожному можливому значенню x.

3.5.Оцінка генератора трафіку

Дані розрахунки проводилися на двох уявних колекціях мережевого трафіку. Різні характеристики цих двох колекцій наведені в таблиці 3.5. Як видно з таблиці, дві колекції трафіку мають різні характеристики; у них різний час збору, і в трафіку EMC майже втричі більше потоків, але потоки в трафіку EMC набагато коротші, ніж потоки BGU, і містять набагато менше пакетів. [23] Крім того, хоча трафік EMC містить приблизно половину кількості послідовності порівняно з трафіком BGU, він містить у шість разів більше комунікаційних пар IP, що ускладнює трафік, принаймні для агрегації на основі IP.

Таблиця 3.2. — Різні характеристики трафіку, використовуваного для експериментів.

	BGU	**EMC**
Розмір	20.1 Гб	2.79 Гб
Кількість пакетів	193 50 505	8 700 370
Кількість потоків	190 462	549 350
Source IPs	1 329	633
Destination IPs	1 309	435
Source та Destination	3 322	24 760
Мінімальний час	1.10.2013 1:05:57	13.06.2013 07:38:27
Максимальний час	2.10.2013 1:11:07	13.06.2017 12:59:14
Розмір послідовності	57 721 потоків	22 367 потоків

3.5.1.Результати оцінок

У наборі наших оцінок зосередимося на оцінці та адаптації моделі кластеризації. Ми розглянули три різні конфігурації кластеризації, змінивши кількість кластерів для кожної конфігурації (K = 100, 500, 1000). Попередньо оброблений трафік BGU був кластеризований відповідно до кожної з цих конфігурацій, використовуючи k-means реалізацію scikit-learn. Для кожної оцінюваної конфігурації послідовну модель потім навчали за допомогою глобальної моделі Маркова і можливо використовуватиметься для генерації зразків трафіку.

Ми оцінювали якість структури кластерів за допомогою оцінки силуету, як описано на Рисунок 3.4. Нагадаємо, що більш високі показники вказують на чіткіший поділ на кластери, тому результати показують, що для досліджуваних конфігурацій збільшення кількості кластерів (K) покращується структура кластерів, однак це відбувається за ціною істотно збільшуваних модельних часів навчання, як видно на Рисунку 3.5.

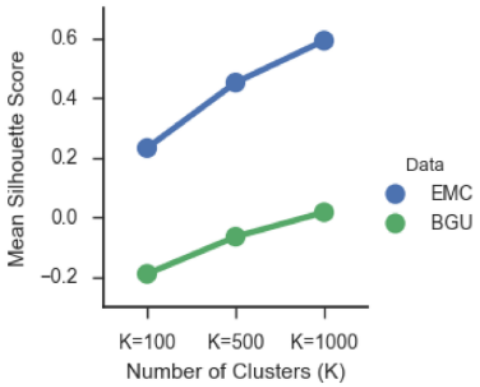


Рисунок 3.4.Середні показники силуету проти кількості кластерів.

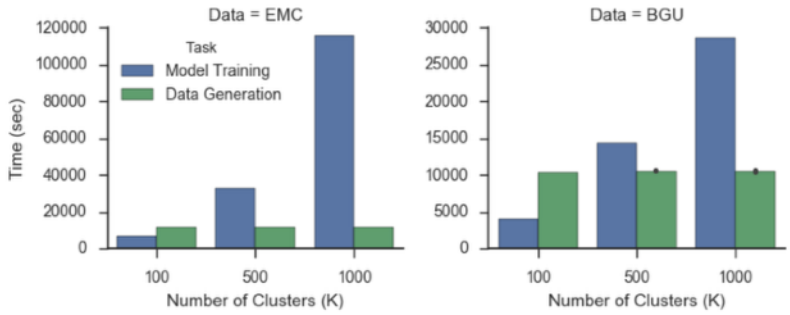


Рисунок 3.5. Час виконання порівняно з кількістю кластерів для модельних завдань (синій) та формування даних (зелений).

Далі ми мали на меті зрозуміти, яка з особливостей найбільше вплинула на процес кластеризації. Для цього завдання ми використовували класифікатор випадкових дерев, який зазвичай використовується для подібних завдань. Для кожної досліджуваної конфігурації кластеризації ми підготували модель Random Forest для класифікації попередньо обробленого трафіку на кластери, де цільовою змінною (clusterID) є ідентифікатор кластеру, отриманий у фазі кластеризації. Потім ми витягли топ-10 найвпливовіших особливостей у навченій моделі Random Forest (вимірюється на основі інформаційного посилення різних функцій). У таблиці 3 наведено інформацію про топ-10 найвпливовіших функцій для кожної дослідженої конфігурації на основі трафіку BGU. Перші шість особливостей у всіх конфігураціях базуються на часі дня, коли відбувся потік (функція денного часу). Функція TCP SeqID також є домінуючою для 500 та 1000 кластерів, тоді як для 100 кластерів переважна функція розміру вікна TCP. [28] День тижня, в якому відбувся потік (особливість тижня, день), також впливає при використанні 100 або 500 кластерів. Досить схожі результати були отримані для набору даних EMC.

Інший спосіб оцінити якість результатів кластеризації - перевірити, наскільки збережені ймовірності різних ознак. Для кожної з розглянутих конфігурацій кластеризації та для кожної з 197 числових ознак, що відображаються в даних, ми використовували двопробний тест KS, щоб перевірити, чи розподіл функції у вихідному та створеному трафіку був однаковим на рівні значущості 0,01. Аналогічним чином ми використовували тест AD на двох зразках, щоб перевірити, чи збережено розподіл семи номінальних ознак. На Рисунок 3.6 показані результати, отримані для кожної тестованої конфігурації кластера. Сто кластерів працювали найкраще, зберігаючи розподіл більшості функцій. Використання більше 100 кластерів послаблює результати, ймовірно, тому, що рідкісні кластери можуть не відображатися в генерованому трафіку, так що відсутність можливих значень для членів цих кластерів призводить до зміни статистики, але навіть для 1000 кластерів, щонайменше, 50% рис зберігається.

Rank	K=100	K=500	K=1000
1	day time last	day time min	day time min
2	day time min	day time sum	day time mean
3	day time sum	day time last	day time first
4	day time first	day time first	day time sum
5	day time max	day time max	day time max
6	day time mean	day time mean	day time last
7	TCP win size sum	IP ttl sum	TCP seqID first
8	week day first	TCP seqID mean	TCP seqID sum
9	TCP win size thirdQ	week day sum	TCP seqID last
10	packet size mean	TCP seqID last	TCP seqID mean

Рисунок 3.6. Топ-10 найвпливовіших функцій для кластеризації трафіку BGU з використанням різної кількості кластерів (K). (Особливості, витягнуті з одного і того ж атрибута, відображаються в одному кольорі.)

Для того, щоб оцінити, наскільки добре збережені шаблони послідовностей у генерованому трафіку, ми вимірювали невдачі для n-грамів з трьома різними розмірами (n = 2, 3, 4) та обчислювали різниці між неприємностями у вихідному та генерованому трафіку. Взагалі важче зберегти послідовності для довгих підрядів, тому заплутаність та їх відмінності зазвичай збільшуються з n. Як показано на Рисунку 3.7, 100 кластерів призводять до найменшої різниці між n-гранними невдачами у вихідних та згенерованих даних, і тому ця кількість кластерів є найкращою конфігурацією кластеризації для збереження шаблонів послідовностей. Це розумно, оскільки зменшення кількості параметрів полегшує збереження розподілу. [26]Використання 100 кластерів призводить до найкращих показників як з точки зору якості генерованого трафіку, так і часу роботи.

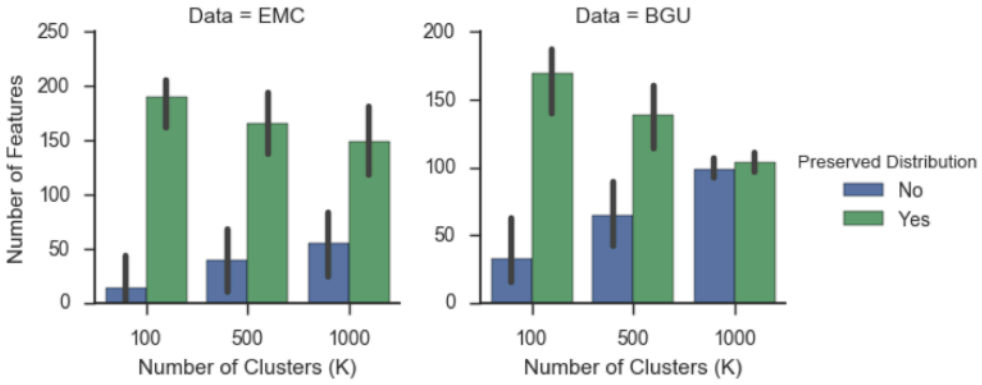


Рисунок 3.7. Кількість функцій, розподіл яких однаковий (зелений) або різний (синій) як у вихідному, так і в генерованому трафіку, згідно з тестами KS та AD (α = 0,01) проти кількості кластерів.

3.6.Висновки

У цьому дослідженні ми представили розрахунки прототипу NTG для генерації мережевого трафіку, яка зберігає як розподіл множинних характеристик трафіку, так і послідовності шаблонів на рівні потоку. Цей метод є особливо потужним для різноманітного мережевого трафіку; у цьому сценарії схеми, які аналітику буде важко витягти та кодувати вручну через їх складний характер, успішно видобуватимуться NTG. Експерименти, проведені на двох уявних колекціях мережевого трафіку, продемонстрували, що трафік, який прирустимо може бути згенерований за допомогою NTG, збереже безліч характеристик вихідного трафіку, а також послідовності моделей діяльності мережі.

Це дослідження демонструє, що можна використовувати обмежену кількість доступного мережевого трафіку для створення масштабованого, регульованого та репрезентативного мережевого трафіку. Згенерований трафік може використовуватися для досліджень, розробки та тестування систем. Отримані результати показують, що існує компроміс між збереженням розподілів основних характеристик трафіку та збереженням послідовності мережевих дій.

Можна припустити, що витяг шаблонів послідовностей мережевих дій може істотно сприяти ідентифікації мережевих атак, планується вивчити цей напрямок більше детально у майбутньому, провести експеременти.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ

4.1.Постановка задачі проектування

Проводиться оцінка основних параметрів, що вплывають на написання скрипту для отримання мережевих метаданих та їх відбитків таких як JA3 та HASSH з файлів захоплення пакетів(.pcap) або живого мережевого трафіку. Розроблена програмна реалізація здійснена за допомогою мови високого рівня Python та відповідного середовища розробки від компанії JetBrains - PyCharm.

4.2.Обґрунтування функцій та параметрів програмного продукту

Головна функція F_0 – скрипт для отримання мережевих метаданних. Виходячи з конкретної мети, виділимо основні функції прогрманого продукту:

- F_1 – вибір джерела вхідного трафіку;
- F_2 – вибір протоколів, що підтримуються;
- F_3 – вибір кількості параметрів для виведення даних.

Для кожної з функцій можна співставити декілька варіантів реалізації:

- Функція F_1:1)поточний мережевий трафік; 2)трафік з файлу захопленних пакетів
- Функція F_2:1)протокол SSL/TLS; 2)протокол SSH; 3)протокол RDP; 4)протокол HTTP
- Функція F_3:1)досить стандартного виведення даних(найменшої кількості параметрів); 2)досить розширеного виведення даних(більшою ніж найменша кількість параметрів)

Виходячи з наведених варіантів маємо морфологічну карту системи (Рисунок 4.1.2), на основі цієї карти будуємо позитивно-нагативну матрицю варіантів основних функцій:

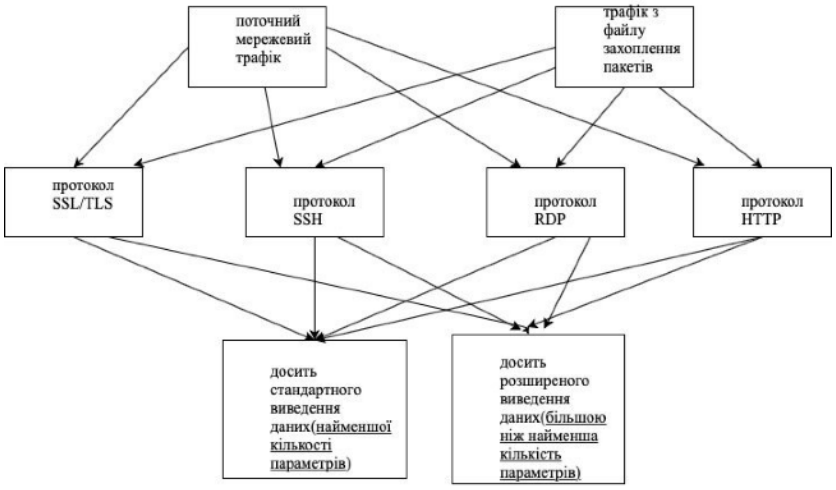


Рисунок 4.1. Морфологічна карта.

Спираючись на карту була побудована позитивно-негативна матриця(табл. 4.1).

Таблиця 4.1 Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Недоліки	Переваги
F1	*A*	Потреба додаткового аналізу відповідного трафіку по мірі його надходження	Більш проста можливість отримання трафіку
	B	Функціонал зчитування трафіку з файлу захоплених пакетів	Відсутність потреби аналізу по мірі надходження(фіксований об'єм даних)
F2	*A*	Складна здатність масштабування. Поверхневий аналіз.	Низькі часові витрати для обробки даних
	B	Розширені часові витрати для аналізу. Потреба велких ресурсів	Можливість глибокого аналізу пакетів. Легко масштабуються
F3	*A*	Недостатня кількість функціоналу	Низька ціна
	B	Висока вартість	Можливість використання більшої кількості функціоналу

Для характеристики прототипу програмного додатку використовуємо параметри X1 – X5. На основі даних, що представлені у літературі, визначаємо мінімальні, середні отримуванні та максимально допустимі значення(табл. 4.2)

Таблиця 4.2 Система параметрів додатку

Назва параметру	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Продуктивність мови програмування	X1	Оп/мс	4000	8000	16000
Орієнтовна кількість програмного коду	X2	рядків	4000	1500	1000
Середня ціна при аналізі за певним критерієм	X3	Кількість автомобілів	5	3	0
Середнє навантаження	X4	Кількість годин на добу	0	10	24

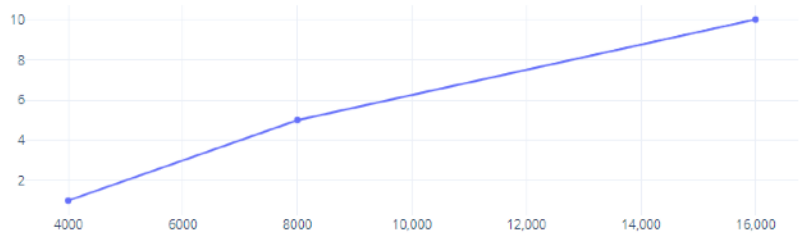


Рисунок 4.2 . Бальна оцінка продуктивності мови програмування

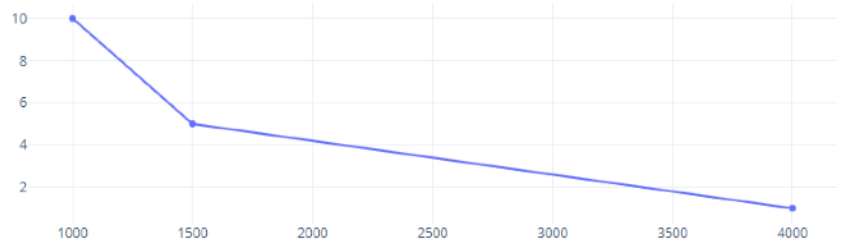


Рисунок 4.3. Бальна оцінка орієнтовної кількісті програмного коду

Рисунок 4.4. Бальна оцінка середної ціни при аналізі за певним критерієм

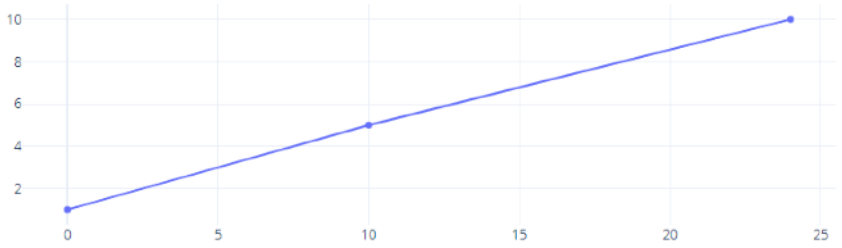


Рисунок 4.5. Бальна оцінка середнього навантаження

Вагомість параметрів оцінюється за допомогою методів попарного зрівняння. Ранги варіюються від 1 до 5.Результати наведені в табл. 4.3-4.4 –

Таблиця 4.3 Результат оцінки параметрів

Познач. параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
	1	2	3	4	5	6	7			
X1	3	2	4	3	3	2	3	20	2.5	6.25
X2	4	4	3	4	4	4	4	27	9.5	90.25
X3	1	3	2	2	2	3	1	14	-3.5	12.25
X4	2	1	1	1	1	1	2	9	-8.5	72.25
Разом	10	10	10	10	10	10	10	70	0	181

За найбільший ранг приймаємо 4

Таблиця 4.4 Попарне зрівняння параметрів

Параметри	Експерти							Підсумкова	Числове
	1	2	3	4	5	6	7	оцінка	значення
X1,X2	<	<	>	<	<	<	<	<	0.5
X1,X3	>	<	>	>	>	<	>	>	1.5
X1,X4	>	>	>	>	>	>	>	>	1.5
X2,X3	>	>	>	>	>	>	>	>	1.5
X2,X4	>	>	>	>	>	>	>	>	1.5
X3,X4	>	>	>	>	>	>	>	>	1.5

Визначемо коефіцієнт конкординації:

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 181}{7^2\left(4^3 - 4\right)} = 0.72 > W_k = 0{,}67.$$

Так як коефіцієнт конкординації більше нормативного, результати вважають достовірними. Розрахунок вагомості параметрів наведено в таблиці 4.5

Таблиця 4.5 Розрахунок вагомості параметрів

Параметри	Параметри				Перший крок		Другий крок		Третій крок	
	X1	X2	X3	X4	b_i	Кві	b_i^1	$\hat{E}_{\hat{a}^3}^1$	b_i^1	$\hat{E}_{\hat{a}^3}^1$
X1	1.0	0.5	1.5	1.5	5	0.3	16.75	0.2757	60.75	0.273495
X2	1.5	1.0	1.5	1.5	5.5	0.33	22	0.36214	80.125	0.36072
X3	0.5	0.5	1.0	1.5	3.5	0.21	12.5	0.205761	46.125	0.207653
X4	0.5	0.5	0.5	1.0	2.5	0.15	9.5	0,1563	35.125	0.158132
Всього:					16.5	1	60.75	1	222.125	1

Таблиця 4.6 Розрахунок показників рівня якості варіантів реалізації

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	а	1	8	0.276	2.88
F2	а	9000	4.2	0.36	1.1592
	б	12000	6.6	0.36	1.8216
F3	а	1500	5	0.207	0.78
	б	3000	3.3	0.207	0.5148
F4	а	8	3.3	0.158	0.6765

Обрахуємо коефіцієнти якості кожного з варіантів розробки: -

$K_{K1} = 2.28 + 1.15 + 0.78 + 0.6765 = 4.88$

$K_{K2} = 2.28 + 1.82 + 0.78 + 0.6765 = 5.56$

$K_{K3} = 2.28 + 1.15 + 0.51 + 0.6765 = 4.62$

$K_{K4} = 2.28 + 1.82 + 0.52 + 0.6765 = 5.29$

Відповідно до розрахунків, кращим серед варіантів є останній, що має вищий коефіцієнт технічного рівня.

4.3.Економічний аналіз варіантів розробки ПП.

Для оцінки трудомісткості розробки спочатку проведемо розрахунок трудомісткості. Усі варіанти мають наступні основні завдання:

1. Розробка проекту додатку;

2. Розробка оболонки програмного продукту;

Також кожний з варіантів має два додаткових завдання, які є реалізаціями розгалужених варіантів розробки незалежного модуля. Далі наведено варіанти додаткових завдань

3.1) вибір протоколу SSL/TSL

3.2)вибір протоколу SSH

4.1) стандартне виведення даних

4.2) розширене виведення даних

В варіанті 1 присутні наступні додаткові завдання під номерами 3.1 та 4.1

В варіанті 2 присутні наступні додаткові завдання під номерами 3.2 та 4.1

В варіанті 3 присутні наступні додаткові завдання під номерами 3.1 та 4.2

В варіанті 4 присутні наступні додаткові завдання під номерами 3.2 та 4.2

Розробка проекту додатку за ступенем новизни належить до групи А, розробка оболонки програмного продукту – до групи Б. За складністю методи, що необхідно імплементувати для пункту 1 належать до групи 1, для другого пункту – група три.

Розглянемо завдання послідовно, починаючи з першого. Орієнтуючись на норми часу для завдань із заданим степним новизни та певною групою складності, трудомісткість дорівнює $T_P = 90$ людино-днів. Для цього застосуємо поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $К_П = 1.7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації для всіх завдань рівний 1: $К_{СК} = 1$. Через використання стандартних модулів та бібліотек при розробці, врахуємо даний факт, як коефіцієнта $К_{СТ} = 0.8$. Результатом є загальна трудомісткість програмування першого завдання, що розраховується, як:

$T_1 = 90 \cdot 1.7 \cdot 0.8 = 122.4$ людино-днів.

Для другого завдання параметри будуть рівні: $К_П = 0.9$, $К_{СК} = 1$, $К_{СТ} = 0.8$:

$T_2 = 27 \cdot 0.9 \cdot 0.8 = 19.44$ людино-днів.

Для завдання 3.1 (степінь новизни В, група складності – 3), тобто $T_P = 12$ людино-днів, $К_П = 0,6$; $К_{СК} = 1$; $К_{СТ} = 0.8$:

$T_{3.1} = 12 \cdot 0,6 \cdot 1 \cdot 0.8 = 5.76$ людино-днів.

Для завдання 3.2(степінь новизни В, складність алгоритмів 2) , тобто $T_P = 19$ людино-днів, $К_П = 0,72$; $К_{СК} = 1$; $К_{СТ} = 0.8$

$T_{3.2}$ = 19·0,72·1· 0.8=10,944 людино-днів.

Для завдання 4.1 (степінь новизни Б, група складності – 2), тобто T_P=27 людино-днів,$К_П$ = 1.08; $К_{СК}$ = 1; $К_{СТ}$ = 0.8:

$T_{4.1}$= 27·1,08·1· 0.8 = 17.28 людино-днів.

Для завдання 4.2(степінь новизни А, група складності — 1), тобто T_P=90 людино-днів, $К_П$ = 1,7 , $К_{СК}$ = 1; $К_{СТ}$ = 0.8.

$T_{4.2}$ = 90·2,84 ·1·0.8 =204,48людино-днів

Сумуємо трудомісткість для кожного із завдань для кожного з обраних варіантів реалізації програми для отримання кінцевої трудомісткості:

T_I = (122.4 + 19.44 + 5.76 + 17.28) · 8 = 1319,04 людино-годин;

T_{II} = (122.4 + 19.44 + 10.95 + 17.28) · 8 = 1360.56 людино-годин;

T_{III} = (122.4 + 19.44 + 5.76 + 204.48) · 8 = 2816,64 людино-годин;

T_{IV} = (122.4 + 19.44 + 10.95 + 204.48) · 8 = 2858.16 людино-годин;

Більшу трудомісткість має варіант IV.

Для розробки необхідно задіяти два програмісти з окладом 5321 грн. та одного фінансового аналітика з окладом 7643грн. Визначимо погодинну зарплату

$$СЧ = \frac{5321 + 5321 + 7643}{3 \cdot 21 \cdot 8} = 36.28\ грн\ .$$

Зарплата розробників для кожного варіанту становить:

I. $С_{ЗП}$ = 36.28 · 1319.04 · 1.2 = 57425,72 грн.

II. $С_{ЗП}$ = 36.28 · 1360.56 · 1.2 = 59233,34 грн.

III. $С_{ЗП}$ = 36.28 · 2816.64 · 1.2 = 122625,23 грн.

IV. $С_{ЗП}$ = 36.28 · 2858.16 · 1.2 = 124345,78 грн.

Вирахуємо відрахування на єдиний соціальний внесок (ЄСВ), що становить 22,00%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 57425,72 \cdot 0.22 = 12633.65$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 59233,34 \cdot 0.22 = 13031.33$ грн.

III. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 122625,23 \cdot 0.22 = 26977.55$ грн.

IV. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 124345,78 \cdot 0.22 = 27356.07$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 5321 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 5321 \cdot 0.2 = 12770.4$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 12770.4 \cdot (1 + 0.2) = 15324.48$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 15324.48 \cdot 0,22 = 3371.39$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 12222 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 12222 = 3513.83$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Додатково врахуємо кошти на ремонт та профілактику:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 12222 \cdot 0.05 = 702.77$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК з розрахунком на один рік буде наступним:

$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = 1706.4$ годин, де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Необхідно також зазначити витрати на оплату електроенергії:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706.4 \cdot 0,833 \cdot 0,9 \cdot 1,75 = 2239,65$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 12222 \cdot 0,67 = 8188.74$ грн.

У результаті, експлуатаційні витрати на рік становитимуть:

$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$

$С_{ЕКС} = 15324.48 + 3371.39 + 3513.83 + 702.77 + 2239,65 + 8188.74 = 30180.86$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 30180.86 / 1706,4 = 17.68$ грн/час.

Так як в обидвох варіантах всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації вираховуються за формулами:

$С_М = С_{М\text{-}Г} \cdot Т$

I. $С_М = 17.68 \cdot 1319,04 = 23320.62$ грн.;

II. $С_М = 17.68 \cdot 1360.56 = 24054.7$ грн.;

I. $С_М = 17.68 \cdot 2816,64 = 49798.2$ грн.;

II. $С_М = 17.68 \cdot 2858.16 = 50532.26$ грн.;

Накладні витрати складатимуть 67% від заробітної плати: $С_Н = С_{ЗП} \cdot 0,67$

I. $С_Н = 57425,72 \cdot 0,67 = 38475.23$ грн.;

II. $С_Н = 59233,34 \cdot 0.67 = 39686.33$ грн.;

I. $С_Н = 122625,23 \cdot 0,67 = 82158.9$ грн.;

II. $С_Н = 124345,78 \cdot 0.67 = 83311.67$ грн.;

Отже, вартість розробки програмного продукту для кожного варіанту становить:$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н}$

I. $С_{ПП} = 57425,72 + 12633.65 + 23320.62 + 38475.23 = 131855.22$ грн.;

II. $С_{ПП} = 59233,34 + 13031.33 + 24054.7 + 39686.33 = 136005.7$ грн.;

I. $С_{ПП} = 122625,23 + 26977.55 + 49798.2 + 82158.9 = 281559.88$ грн.;

II. $С_{ПП} = 124345,78 + 27356.07 + 50532.26 + 83311.67 = 285545.78$ грн.;

4.4.Обґрунтування функцій та параметрів програмного продукту

Вирахуємо коефіцієнт техніко-економічного рівня за наступною формулою: $К_{ТЕРj} = К_{Кj} / С_{Фj}$,

$К_{ТЕР1} = 4.88 / 131855.22 = 3,7 \cdot 10^{-5}$;

$К_{ТЕР2} = 5.56 / 136005.7 = 4,08 \cdot 10^{-5}$;

$К_{ТЕР3} = 4.62 / 281559.88 = 1,64 \cdot 10^{-5}$;

$К_{ТЕР4} = 5.29 / 285545.78 = 1,87 \cdot 10^{-5}$;

Виконавши функціонально-вартісний аналіз програмного комплексу, оптимальною альтернативою для створення програмного продукту є другий варіант реалізації з найвищим показником техніко-економічного рівня якості

$К_{ТЕР2} = 4,08 \cdot 10^{-4}$.

4.5.Висновки

Після виконання функціонально-вартісного аналізу програмного продукту що розроблюється, можна зробити висновок, що другий варіант є найбільш оптимальним для реалізації (через більше значення коефіцієнту технічного рівня). Його показник техніко-економічного рівня якості $КТЕР2 = 4,08 \cdot 10^{-5}$;

Даний варіант імплементації програмного продукту відповідає таким параметрам:

- джерело вхідного трафіку обрати поточний трафік

- протокол SSL/TLS
- використання розширеного формату виведення даних.

Використання розробленого скрипту надає змогу проведення аналізу мережевого трафіку відповідного джерела з розширенимим характеристиками при виведенні.

ВИСНОВКИ

Таким чином формуємо результати даної роботи:

1.Досліджено процес аналіза трафіку.

2.На основі цього дослідження проведено порівняльний аналіз існуючих засобів аналізу трафіку. Реалізовано базовий функіонал пакетного аналізатора для деяких мережевих протоколів з унікальною реалізацією методів створення відбитків.

3.З розробленого аналізатора з’являється потреба моделювання мережевого трафіку відповідно проведено дослідження моделювання трафіку. Базуючись на цьому розроблена та досліджена модель моделювання мережевого трафіку зі збереженням послідовності.

Переваги розробленого пакетного аналізатора, які виходять з використання відповідних методів створення відбитків:

- Можна виявити, контролювати і досліджувати спроби взлому з більш високим рівнем деталізації, ніж джерело IP.

- Виявити приховану ексфільтрацію даних у компонентах набору клієнтських алгоритмів. У цьому випадку спеціально кодований SSH-клієнт може надсилати вихідні дані з надійного середовища до менш надійного середовища в межах серії пакетів SSH_MSG_KEXINIT. У сценарії, подібному до більш відомої ексфільтрації через DNS, дані можуть бути надіслані у вигляді серії спроб, але неповних і незамкнутих підключень до SSH-сервера. Такі спроби не реєструються навіть відомими аналізаторами пакетів або в системах кінцевих точок. Виявлення цього стилю ексфільтрації тепер може бути легко здійснено за допомогою виявлення аномалії або оповіщення для клієнтів SSH з декількома різними помилками.

- Створення додаткового рівеня управління клієнтськими додатками, наприклад, ви можете заблокувати підключення всіх клієнтів до SSH-

сервера, який знаходиться поза затвердженим відомим набором значень hassh.

- Виявлення пристроїв, які мають хеш-файли, які, як відомо, належать до вбудованих систем IOT. Приклади можуть включати камери, мікрофони, кейлоггери, які можна легко приховати від перегляду та спокійного спілкування по зашифрованим каналам.

По завершенню відповідного дослідження планується до вектору розвитку FATT додати реалізацію моделі моделювання мережевого трафіку зі збереженням послідовності. Таким чином розширивши можливості новоствореного продукту не лише аналізом трафіку, а відповідної можливості його генерації.

ПЕРЕЛІК ПОСИЛАНЬ

1.Пакетні сніфери. URL: https://www.inter-nauka.com/uploads/public/15895669733068.pdf (дата звернення 01.06.2020)

2.Mike Cloppert. An Overview Of Protocol Reverse-Engineering. URL: https://digitalforensics.sans.org/blog/2012/07/03/an-overview-of-protocol-reverse-engineering (дата звернення 02.03.2020)

3. Що таке сніфер? URL: https://www.avg.com/en/signal/what-is-sniffer (дата звернення: 13.05.2020)

4. Stephen Northcat, Judy Nowak, Network Security Discovery. 3rd ed. New York: Sams Publishing, 2003. 356 p.

5. Кращі 10 сніферних пакетів та їх інструментів у 2020 URL: https://www.dnsstuff.com/packet-sniffers (дата звернення: 13.05.2020)

6. Orebaugh Angela ,Wireshark network protocol analyzer, 2006. 450 p.

7.The Bro Network Security Monitor. URL: http://www.bro.org/ (дата звернення 07.03 .2020)

8. Wireshark. URL: http://www.wireshark.org/ (дата звернення 17.04 . 2020)

9. Tcpdump. URL: http://www.tcpdump.org/ (дата звернення 25.03 . 2020)

10. Wireshark Display Filter Reference. URL: https://www.wireshark.org/docs/dfref/, (дата звернення 21.04 .2020)

11. Таненбаум Э., Уэзеролл Д. Компьютерные сети. 5-е изд. Питер: СПб., 2012. 960 с.

12. А. Ализар. Доля зашифрованного трафика в интернете выросла в несколько раз. URL: https://xakep.ru/2014/05/15/62500/ (дата звернення 05.04 .2020)

13. Сетевые модели OSI и TCP/IP. URL: http://www.quizful.net/post/osi_tcpip_layers (дата звернення 07.05 .2020)

14. Colasoft Packet Player. URL: http://www.colasoft.com/packet_player/ (дата звернення 07.05 .2020)

15. А.В. Никешин, Н. В. Пакулин, В. З. Шнитман. Тестирование реализаций клиента протокола TLS. том 27, выпуск 2, Москва: ИСП РАН, 2015. 160 стр.

16. SSL/TLS. URL: http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node17.html#SECTION00321400000000000000, (дата звернення 12.04 .2020)

17. Microsoft SMB Protocol and CIFS Protocol Overview, URL: https://msdn.microsoft.com/en-us/library/aa365233(VS.85).aspx, (дата звернення 10.05 .2020)

18. Мэйволд Э. «Межсетевые экраны» // Безопасность сетей: Информация, ИНТУИТ, 2006. URL: http://www.intuit.ru/studies/courses/102/102/lecture/2989, (дата звернення 22.05 .2020)

19. Christopher Parsons. Deep Packet Inspection in Perspective: Tracing its lineage and surveillance potentials. URL: https://www.academia.edu/9603296/Deep_Packet_Inspection_in_Perspective_Tracing_its_lineage_and_surveillance_potentials , (дата звернення 30.05 .2020)

20. Documentation to DCE/RPC, URL: http://www.dcerpc.org/documentation/, (дата звернення 30.05 .2020)

21. Oink: a Collaboration of C/C++ Tools for Static Analysis and Source-to-Source Transformation. URL: http://daniel-wilkerson.appspot.com/oink/index.html/, (дата звернення 01.06 .2020)

22. Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA). Montreal, QC, Canada, 17-19 Oct. 2016, IEEE. 410p.

23. Paul Barford, Mark Crovella, Paul Barford, and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. New York, USA, 1998. 160p.

24. Alessio Botta, Alberto Dainotti, and Antonio Pescape. Do you trust your software-based traffic generator? Montreal, QC, Canada. IEEE Communications Magazine, 48(9): sep 2010. 165p.

25. Joel Sommers and Paul Barford. Self-configuring network traffic generation. In Proceedings of the 4th ACM SIGCOMM conference on Internet measurement . New York, USA, 2004.108p.

26. J. A. Hartigan and M. A. Wong. Algorithm AS 136: A K-Means Clustering Algorithm. New York: Wiley, 1979. 108p.

27. Xiaoming Zhou and Piet Van Mieghem. Reordering of IP Packets in Internet. 5th ed. France: PAM, 2004, 246p.

28. Arjuna Sathiaseelan and Tomasz Radzik. Improving the performance of TCP in the case of packet reordering. France: IEEE. 2004. 73p

ДОДАТОК А НАУКОВІ ДОСЯГНЕННЯ

Список наукових публікацій

студентки гр.КА-65, **Авксентьєвої І.О.**

Статті

1. Кухарєв С.О., Авксентьєва І.О. Пакетні сніфери // Міжнародний науковий журнал "Інтернаука". — 2020. — №7.

ДОДАТОК Б ЛІСТИНГ ПРОГРАМИ

```
...
def parse_cmd_args():
    """parse command line arguments"""
    desc = """A python script for extracting network fingerprints"""
    parser = argparse.ArgumentParser(description=(desc))
    helptxt = "pcap file to process"
    parser.add_argument('-r', '--read_file', type=str, help=helptxt)
    helptxt = "directory of pcap files to process"
    parser.add_argument('-d', '--read_directory', type=str, help=helptxt)
    helptxt = "listen on interface"
    parser.add_argument('-i', '--interface', type=str, help=helptxt)
    helptxt = "protocols to fingerprint. Default: all"
    parser.add_argument(
        '-fp',
        '--fingerprint',
        nargs='*',
        default='all',
        choices=['tls', 'ssh', 'rdp', 'http', 'gquic'],
        help=helptxt)
    helptxt = "a dictionary of {decode_criterion_string: decode_as_protocol} \
        that is used to tell tshark to decode protocols in situations it \
        wouldn't usually."
    parser.add_argument(
        '-da', '--decode_as', type=json.loads, default=DECODE_AS, help=helptxt)
    helptxt = "BPF capture filter to use (for live capture only).'"
    parser.add_argument(
        '-f', '--bpf_filter', type=str, default=CAP_BPF_FILTER, help=helptxt)
    helptxt = "log the output in json format"
    parser.add_argument(
```

```
        '-j', '--json_logging', action="store_true", help=helptxt)
    helptxt = "specify the output log file. Default: fatt.log"
    parser.add_argument(
        '-o', '--output_file', default='fatt.log', type=str, help=helptxt)
    helptxt = "save the live captured packets to this file"
    parser.add_argument(
        '-w', '--write_pcap', default=None, type=str, help=helptxt)
    helptxt = "print the output"
    parser.add_argument(
        '-p', '--print_output', action="store_true", help=helptxt)
    return parser.parse_args()


def main():
    """intake arguments from the user and extract RDP client fingerprints."""
    global DISPLAY_FILTER
    args = parse_cmd_args()
    setup_logging(args.output_file)
    fingerprint = args.fingerprint
    jlog = args.json_logging
    pout = args.print_output

    pp = ProcessPackets(fingerprint, jlog, pout)

    # Process PCAP file
    if args.read_file:
        cap = pyshark.FileCapture(
            args.read_file,
            display_filter=DISPLAY_FILTER,
            keep_packets=False,
            decode_as=args.decode_as)
```

```
    try:
        for packet in cap:
            pp.process(packet)
        cap.close()
        cap.eventloop.stop()
    except Exception as e:
        print('Error: {}'.format(e))
        pass
# Process directory of PCAP files
elif args.read_directory:
    files = [f.path for f in os.scandir(args.read_directory)
             if not f.name.startswith('.') and not f.is_dir() and
             (f.name.endswith(".pcap") or f.name.endswith(".pcapng") or
             f.name.endswith(".cap"))]
    for file in files:
        cap = pyshark.FileCapture(
            file,
            display_filter=DISPLAY_FILTER,
            keep_packets=False,
            decode_as=args.decode_as)
        try:
            for packet in cap:
                pp.process(packet)
            cap.close()
            cap.eventloop.stop()
        except Exception as e:
            print('Error: {}'.format(e))
            pass

# Capture live network traffic
elif args.interface:
```

```
    if args.write_pcap:
        DISPLAY_FILTER = None
    # TODO: Use a Ring Buffer (LiveRingCapture), when the issue is fixed:
    # https://github.com/KimiNewt/pyshark/issues/299
    cap = pyshark.LiveCapture(
        interface=args.interface,
        decode_as=args.decode_as,
        display_filter=DISPLAY_FILTER,
        bpf_filter=args.bpf_filter,
        output_file=args.write_pcap)
    try:
        cap.apply_on_packets(pp.process)
    except (KeyboardInterrupt, SystemExit):
        print("Exiting..\nBYE o/\n")


if __name__ == '__main__':
    main()
```

ДОДАТОК В РЕЗУЛЬТАТИ РОБОТИ

```
$ python3 fatt.py -i en0 --print_output --json_logging
192.168.1.10:59565 -> 192.168.1.3:80 [HTTP]
hash=598c34a2838e82f9ec3175305f233b89 userAgent="Spotify/109600181
OSX/0 (MacBookPro14,3)"
192.168.1.10:59566 -> 13.237.44.5:22 [SSH]
hassh=ec7378c1a92f5a8dde7e8b7a1ddf33d1 client=SSH-2.0-OpenSSH_7.9
13.237.44.5:22 -> 192.168.1.10:59566 [SSH]
hasshS=3f0099d323fed5119bbfcca064478207 server=SSH-2.0-
babeld-80573d3e
192.168.1.10:59584 -> 93.184.216.34:443 [TLS]
ja3=e6573e91e6eb777c0933c5b8f97f10cd serverName=example.com
93.184.216.34:443 -> 192.168.1.10:59584 [TLS]
ja3s=ae53107a2e47ea20c72ac44821a728bf
192.168.1.10:59588 -> 192.168.1.3:80 [HTTP]
hash=598c34a2838e82f9ec3175305f233b89 userAgent="Spotify/109600181
OSX/0 (MacBookPro14,3)"
192.168.1.10:59601 -> 216.58.196.142:80 [HTTP]
hash=d6662c018cd4169689ddf7c6c0f8ca1b userAgent="curl/7.54.0"
216.58.196.142:80 -> 192.168.1.10:59601 [HTTP]
hash=c5241aca9a7c86f06f476592f5dda9a1 server=gws
192.168.1.10:54387 -> 216.58.203.99:443 [QUIC] UAID="Chrome/
74.0.3729.169 Intel Mac OS X 10_14_5" SNI=clientservices.googleapis.com
AEAD=AESG KEXS=C255
```

ДОДАТОК Г РЕЗУЛЬТАТИ РОБОТИ (JSON FORMAT)

{"timestamp": "2020-05-28T03:41:25.415086", "sourceIp": "192.168.1.10", "destinationIp": "192.168.1.3", "sourcePort": "59565", "destinationPort": "80", "protocol": "http", "http": {"requestURI": "/DIAL/apps/com.spotify.Spotify.TVv2", "requestFullURI": "http://192.168.1.3/DIAL/apps/com.spotify.Spotify.TVv2", "requestVersion": "HTTP/1.1", "requestMethod": "GET", "userAgent": "Spotify/109600181 OSX/0 (MacBookPro14,3)", "clientHeaderOrder": "connection,accept_encoding,host,user_agent", "clientHeaderHash": «598c34a2838e82f9ec3175305f233b89"}}
{"timestamp": «2020-05-28T03:41:26.099574", "sourceIp": "13.237.44.5", "destinationIp": "192.168.1.10", "sourcePort": "22", "destinationPort": "59566", "protocol": "ssh", "ssh": {"server": "SSH-2.0-babeld-80573d3e", "hasshServer": "3f0099d323fed5119bbfcca064478207", "hasshServerAlgorithms": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256;chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes192-cbc,aes128-cbc;hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib,zlib@openssh.com", "hasshVersion": "1.0", "skex": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256", "seastc": "chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes192-cbc,aes128-cbc", "smastc": "hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1", "scastc": "none,zlib,zlib@openssh.com", "slcts": "[Empty]", "slstc": "[Empty]", "seacts": "chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes192-

cbc,aes128-cbc", "smacts": "hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1", "scacts": "none,zlib,zlib@openssh.com", "sshka": "ssh-dss,rsa-sha2-512,rsa-sha2-256,ssh-rsa"}}
{"timestamp": "2020-05-28T03:41:26.106737", "sourceIp": "192.168.1.10", "destinationIp": "13.237.44.5", "sourcePort": "59566", "destinationPort": "22", "protocol": "ssh", "ssh": {"client": "SSH-2.0-OpenSSH_7.9", "hassh": "ec7378c1a92f5a8dde7e8b7a1ddf33d1", "hasshAlgorithms": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c;chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com;umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1;none,zlib@openssh.com,zlib", "hasshVersion": "1.0", "ckex": "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256,diffie-hellman-group14-sha1,ext-info-c", "ceacts": "chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com", "cmacts": "umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1", "ccacts": "none,zlib@openssh.com,zlib", "clcts": "[Empty]", "clstc": "[Empty]", "ceastc": "chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-

gcm@openssh.com,aes256-gcm@openssh.com", "cmastc": "umac-64-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-64@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1", "ccastc": "none,zlib@openssh.com,zlib", "cshka": "rsa-sha2-512-cert-v01@openssh.com,rsa-sha2-256-cert-v01@openssh.com,ssh-rsa-cert-v01@openssh.com,rsa-sha2-512,rsa-sha2-256,ssh-rsa,ecdsa-sha2-nistp256-cert-v01@openssh.com,ecdsa-sha2-nistp384-cert-v01@openssh.com,ecdsa-sha2-nistp521-cert-v01@openssh.com,ssh-ed25519-cert-v01@openssh.com,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,ssh-ed25519"}}
{"timestamp": "2020-05-28T03:41:36.762811", "sourceIp": "192.168.1.10", "destinationIp": "93.184.216.34", "sourcePort": "59584", "destinationPort": "443", "protocol": "tls", "tls": {"serverName": "example.com", "ja3": "e6573e91e6eb777c0933c5b8f97f10cd", "ja3Algorithms": "771,49200-49196-49192-49188-49172-49162-159-107-57-52393-52392-52394-65413-196-136-129-157-61-53-192-132-49199-49195-49191-49187-49171-49161-158-103-51-190-69-156-60-47-186-65-49170-49160-22-10-255,0-11-10-13-16,29-23-24,0", "ja3Version": "771", "ja3Ciphers": "49200-49196-49192-49188-49172-49162-159-107-57-52393-52392-52394-65413-196-136-129-157-61-53-192-132-49199-49195-49191-49187-49171-49161-158-103-51-190-69-156-60-47-186-65-49170-49160-22-10-255", "ja3Extensions": "0-11-10-13-16", "ja3Ec": "29-23-24", "ja3EcFmt": "0"}}
{"timestamp": "2020-05-28T03:41:36.920935", "sourceIp": "93.184.216.34", "destinationIp": "192.168.1.10", "sourcePort": "443", "destinationPort": "59584", "protocol": "tls", "tls": {"ja3s": "ae53107a2e47ea20c72ac44821a728bf", "ja3sAlgorithms": "771,49199,65281-0-11-16", "ja3sVersion": "771", "ja3sCiphers": "49199", "ja3sExtensions": "65281-0-11-16"}}
{"timestamp": "2019-05-28T03:41:37.487609", "sourceIp": "192.168.1.10", "destinationIp": "192.168.1.3", "sourcePort": "59588", "destinationPort": "80",

"protocol": "http", "http": {"requestURI": "/DIAL/apps/
com.spotify.Spotify.TVv2", "requestFullURI": "http://192.168.1.3/DIAL/apps/
com.spotify.Spotify.TVv2", "requestVersion": "HTTP/1.1", "requestMethod":
"GET", "userAgent": "Spotify/109600181 OSX/0 (MacBookPro14,3)",
"clientHeaderOrder": "connection,accept_encoding,host,user_agent",
"clientHeaderHash": "598c34a2838e82f9ec3175305f233b89"}}
{"timestamp": "2020-05-28T03:41:48.700730", "sourceIp": "192.168.1.10",
"destinationIp": "216.58.196.142", "sourcePort": "59601", "destinationPort":
"80", "protocol": "http", "http": {"requestURI": "/", "requestFullURI": "http://
google.com/", "requestVersion": "HTTP/1.1", "requestMethod": "GET",
"userAgent": "curl/7.54.0", "clientHeaderOrder": "host,user_agent,accept",
"clientHeaderHash": "d6662c018cd4169689ddf7c6c0f8ca1b"}}
{"timestamp": "2020-05-28T03:41:48.805393", "sourceIp": "216.58.196.142",
"destinationIp": "192.168.1.10", "sourcePort": "80", "destinationPort": "59601",
"protocol": "http", "http": {"server": "gws", "serverHeaderOrder":
"location,content_type,date,cache_control,server,content_length",
"serverHeaderHash": "c5241aca9a7c86f06f476592f5dda9a1"}}
{"timestamp": "2020-05-28T03:41:58.038530", "sourceIp": "192.168.1.10",
"destinationIp": "216.58.203.99", "sourcePort": "54387", "destinationPort":
"443", "protocol": "gquic", "gquic": {"tagNumber": "25", "sni":
"clientservices.googleapis.com", "uaid": "Chrome/74.0.3729.169 Intel Mac OS
X 10_14_5", "ver": "Q043", "aead": "AESG", "smhl": "1", "mids": "100",
"kexs": "C255", "xlct": "cd9baccc808a6d3b", "copt": "NSTP", "ccrt":
"cd9baccc808a6d3b67f8adc58015e3ff", "stk":
"d6a64aeb563a19fe091bc34e8c038b0a3a884c5db7caae071180c5b739bca3dd7c
42e861386718982fbe6db9d1cb136f799e8d10fd5a", "pdmd": "X509", "ccs":
"01e8816092921ae8", "scid": "376976b980c73b669fea57104fb725c6"}}

ДОДАТОК Д ТЕСТУВАННЯ ВРАЗЛИВОСТЕЙ CVE-2020-0708 RDP

```
$ python3 fatt.py -r RDP/cve-2019-0708_metasploit_aux.pcap -p -j; cat fatt.log | python -m json.tool
192.168.1.10:39079 -> 192.168.1.20:3389 [RDP]
rdfp=3ba3d115055e593e3550575a36e68153 cookie="mstshash=user0" req_protocols=0x00000000
{
   "destinationIp": "192.168.1.20",
   "destinationPort": "3389",
   "protocol": "rdp",
   "rdp": {
      "channelDefArray": {
         "0": {
            "name": "cliprdr",
            "options": "c0a00000"
         },
         "1": {
            "name": "MS_T120",
            "options": "80800000"
         },
         "2": {
            "name": "rdpsnd",
            "options": "c0000000"
         },
         "3": {
            "name": "snddbg",
            "options": "c0000000"
         },
         "4": {
```

"name": "rdpdr",
"options": "80800000"
}
},
"clientBuild": "2600",
"clientDigProductId":
"000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0",
"clientName": "x1810",
"clientProductId": "1",
"clusterFlags": "09000000",
"colorDepth": "0x0000ca01",
"connectionType": "0",
"cookie": "mstshash=user0",
"desktopHeight": "600",
"desktopWidth": "800",
"earlyCapabilityFlags": "1",
"encryptionMethods": "03000000",
"extEncMethods": "00000000",
"highColorDepth": "0x00000018",
"keyboardFuncKey": "12",
"keyboardLayout": "1033",
"keyboardSubtype": "0",
"keyboardType": "4",
"pad1Octet": "00",
"postbeta2ColorDepth": "0x0000ca01",
"rdfp": "3ba3d115055e593e3550575a36e68153",
"rdfpAlgorithms": "4,8,09000000,03000000,00000000,cliprdr:c0a00000-MS_T120:80800000-rdpsnd:c0000000-snddbg:c0000000-rdpdr:80800000",
"rdfpVersion": "0.3",

```
        "requestedProtocols": "0x00000000",
        "sasSequence": "43523",
        "serialNumber": "0",
        "supportedColorDepths": "0x00000007",
        "verMajor": "4",
        "verMinor": "8"
    },
    "sourceIp": "192.168.1.10",
    "sourcePort": "39079",
    "timestamp": "2019-05-23T03:51:25.438445"
}
```

ДОДАТОК Е ТЕСТУВАННЯ ВРАЗЛИВОСТЕЙ CVE-2020-0708

POC

$ python3 fatt.py -r RDP/cve-2019-0708_poc.pcap -p -j; cat fatt.log | python -m json.tool
192.168.1.10:54303 -> 192.168.1.20:3389 [RDP] req_protocols=0x00000001


```
{
    "destinationIp": "192.168.1.20",
    "destinationPort": "3389",
    "protocol": "rdp",
    "rdp": {
        "requestedProtocols": "0x00000001"
    },
    "sourceIp": "192.168.1.10",
    "sourcePort": "54303",
    "timestamp": "2019-05-23T18:41:42.572758"
}
```

ДОДАТОК Є ДЕКОДУВАННЯ

```
$ python3 fatt.py -r RDP//cve-2019-0708_poc.pcap -p -j --decode_as
'{"tcp.port==3389": "tls"}'
192.168.1.10:50026 -> 192.168.1.20:3389 [TLS]
ja3=67e3d18fd9dddbbc8eca65f7dedac674 serverName=192.168.1.20
192.168.1.20:3389 -> 192.168.1.10:50026 [TLS]
ja3s=649d6810e8392f63dc311eecb6b7098b

$ cat fatt.log
{"timestamp": "2019-05-23T17:21:56.056200", "sourceIp": "192.168.1.10",
"destinationIp": "192.168.1.20", "sourcePort": "50026", "destinationPort":
"3389", "protocol": "tls", "tls": {"serverName": "192.168.1.20", "ja3":
"67e3d18fd9dddbbc8eca65f7dedac674", "ja3Algorithms":
"771,49196-49195-49200-49199-159-158-49188-49187-49192-49191-49162-49
161-49172-49171-57-51-157-156-61-60-53-47-10-106-64-56-50-19-5-4,0-5-10-
11-13-35-23-65281,29-23-24,0", "ja3Version": "771", "ja3Ciphers":
"49196-49195-49200-49199-159-158-49188-49187-49192-49191-49162-49161
-49172-49171-57-51-157-156-61-60-53-47-10-106-64-56-50-19-5-4",
"ja3Extensions": "0-5-10-11-13-35-23-65281", "ja3Ec": "29-23-24",
"ja3EcFmt": "0"}}
{"timestamp": "2019-05-23T17:21:56.059333", "sourceIp": "192.168.1.20",
"destinationIp": "192.168.1.10", "sourcePort": "3389", "destinationPort":
"50026", "protocol": "tls", "tls": {"ja3s":
"649d6810e8392f63dc311eecb6b7098b", "ja3sAlgorithms":
"771,49192,23-65281", "ja3sVersion": "771", "ja3sCiphers": "49192",
"ja3sExtensions": "23-65281"}}
```

ДОДАТОК Ж МОДЕЛЬ МАРКОВА

Модель Маркова (ММ) є однією з найпоширеніших моделей імовірнісних послідовностей, і вони застосовуються до найрізноманітніших завдань, значною мірою через їх здатність знижувати складність, особливо для довгих послідовностей. Більш конкретно, ММ - це тип ймовірнісної графічної моделі (або байєсівської мережі), яка має ланцюгову топологію

У байєсівській мережі кожна випадкова змінна представлена як вузол у графі, а ребра в графіці спрямовані і представляють імовірнісні залежності між випадковими змінними.

У моделі Маркова ймовірність появи стану в послідовності залежить лише від попередніх m станів послідовності. Модель Маркова другого порядку має найменші обчислювальні витрати, оскільки вона вивчає попередній стан лише при прогнозуванні поточного стану. Ймовірність появи стану як поточного стану в послідовності за моделлю Маркова другого порядку становить:

$$\Pr\left(c_{i+1} = a \mid c_1, c_2, \ldots, c_i\right) \approx \Pr\left(c_{i+1} = a \mid c_i\right)$$

Для формування послідовності кластерів збираються частоти запуску кластерів та кластерні переходи; нехай FD

позначають частоту заданого кластера відповідно до набору кластерних потоків D, так що:

$$\text{StartProb}(a) = F_D\left(c_1 = a\right)$$
$$\text{TransitionProb}(a, b) = \sum_{\forall i} F_D\left(c_{i+1} = b \mid c_i = a\right)$$

ДОДАТОК З МОДЕЛЬ НЕЙРОННОЇ МОВИ

При використанні векторів слів кожне слово ототожнюється з точкою у векторному просторі. Кількість атрибутів (наприклад, m = 64 у наших експериментах) значно менша за розмір словникового запасу. Функція ймовірності виражається як добуток умовних ймовірностей наступного слова з урахуванням попередніх слів (наприклад, використання багатошарової нейронної мережі для прогнозування наступної активності мережі з огляду на попередні дії, в наших експериментах). Ця функція має параметри, які можна ітераційно налаштувати, щоб максимально збільшити ймовірність даних про навчання. Мотивація використання вбудовування слів для дискретних пробілів полягає в тому, що при проектуванні безперервних змінних можна легше домогтися узагальнення, оскільки функція, яка засвоюється, може мати деякі локальні властивості гладкості.

Модель статистичної мови може бути представлена умовною ймовірністю наступного слова, що задається кожним попереднім словом:

$$\hat{p}\left(w_1^T\right) = \prod_{t=1}^{T} \hat{p}\left(w_t \mid w_1^{t-1}\right)$$

де wt - це t-е слово, а підпис підпису

$$w_i^j = \left(w_i, w_{i+1}, \cdots, w_{j-1}, w_j\right)$$

Складність цієї проблеми можна зменшити, скориставшись порядком слів і тим, що слова, ближчі в послідовності, статистично більше залежать. Тому ми прогнозуємо слово, виходячи з контексту попереднього n - 1 слова:

$$\hat{p}\left(w_t \mid w_1^{t-1}\right) \approx \hat{p}\left(w_t \mid w_{t-n+1}^{t-1}\right)$$

Як правило, дослідники використовували триграми (n = 3) і отримували найсучасніші результати. У наших експериментах ми використовували n = 4 для подальшого вдосконалення моделі (на відміну від використання бі-грамів у розглянутій моделі Маркова). Навчання такої масштабної моделі коштує дорого. Однак навчання великих моделей, що враховують більші контексти, дає кращі результати.

Набір тренувань - це послідовність w1,. . . , wt слів wt ∈ V, де словниковий запас V - велика, але кінцева множина. Мета полягає в тому, щоб вивчити хорошу модель

$$f\left(w_t, \ldots, w_{t-n+1}\right) \approx p\left(w_t \mid w_{t-n+1}^{t-1}\right)$$

в тому сенсі, що вона дає велику ймовірність вхідного зразка. Функція f складається з двох частин: 1) Зображення C від будь-якого елемента i V до пов'язаного з ним слова вектор C (i) ∈ Rm. На практиці C представлений a | V | × м матриця вільних параметрів. 2) Функція ймовірності g, яка відображає вхідну послідовність векторів слів у контексті

$$\left(C\left(w_{t-n+1}\right), \ldots, \bar{C}\left(w_{t-1}\right)\right)$$

до умовного розподілу ймовірності над словами у V для наступного слова мас. Вихід g - вектор, і-й елемент якого оцінює

Ймовірність

$$\hat{p}\left(w_{t=i} \mid w_{t-n+1}^{t-1}\right)$$

як на рисунку. Подача або повторювана нейронна мережа з параметрами ω може застосовуватися для функції g. Параметри відображення C - самі функціональні вектори, представлені a | V | × m матриця C, рядок i - слово слова C (i) для слова i. Загальний набір параметрів θ = (C, ω). Навчання досягається шляхом пошуку θ, що максимально збільшує ймовірність журналу вхідного набору даних:

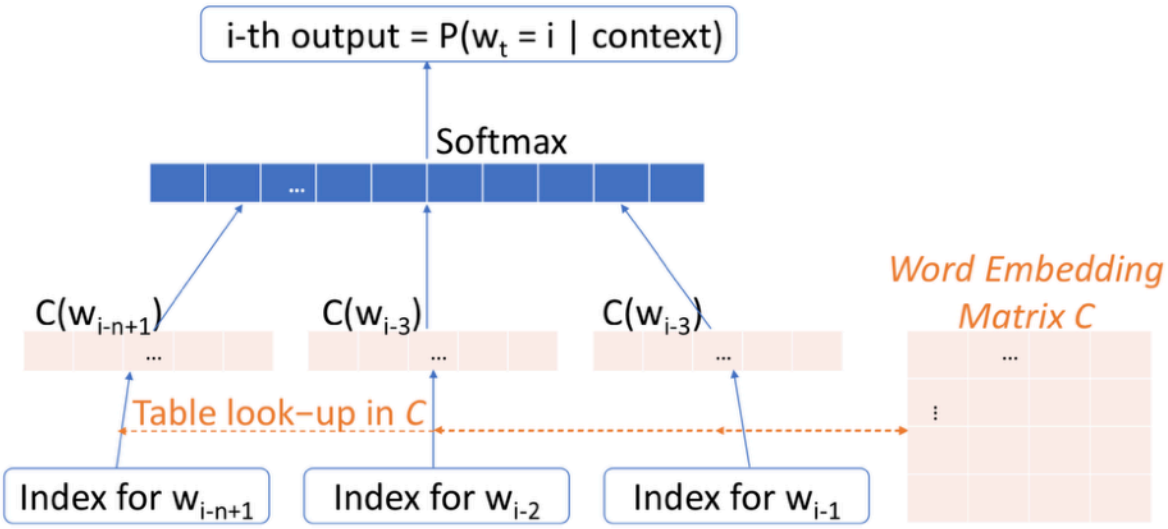


Рисунок 1. Нейронна архітектура

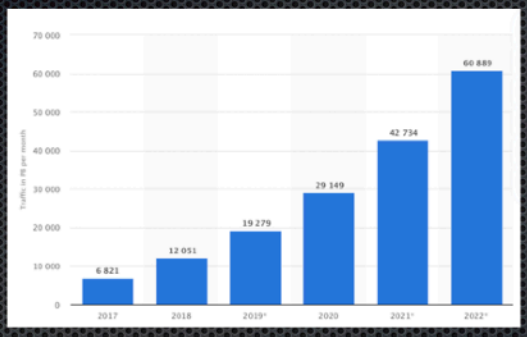
$$f\left(i, w_{t-1}, \cdots, w_{t-n+1}\right) = g\left(i, C\left(w_{t-1}\right), \cdots, C\left(w_{t-n+1}\right)\right)$$

де g - нейронна мережа, а C (i) - і-й слово слова.

$$L = \frac{1}{T} \sum_{t} \left(\log \left(g \left(i, C\left(w_{t-1}\right), \ldots, C\left(w_{t-n+1}\right) \right) \right) \right)$$

ДОДАТОК И ДЕМОНСТРАЦІЙНИЙ МАТЕРІАЛ

Тенденції росту мережевого трафіку



Аналіз трафіку стає все більш потрібним в областях контроля та управління, оптимізації, захисту від шкідливого втручання. Моніторинг трафіку також необхідний, щоб більш ефективно діагностувати і вирішувати проблеми, коли вони постають, таким чином не доводячи різні мережеві сервіси до простоя впродовж тривалого часу. Це обумовлює *актуальність* даної теми. Відповідно до цього ми можемо зрозуміти, що необхідним *об'єктом* дослідження будуть виступати аналізатори та генератори мережевого трафіку, а його *предметом* власне мережевий трафік.

Порівняння існуючих аналізаторов

Параметри	Сніферні інструменти		
	TCPDump	WireShark	Colasoft
Відкритий код	+	+	-
Якими операційними системами підтримується	Linux(WINDum для Windows)	Linux, Windows	Windows
Число підтримки протоколів	TCP/IP	Більш ніж 300	300
Користувацький інтерфейс	CLI	CLI і GUI	GUI
Вартість	Безкоштовно	Безкоштовно	999 $
Libpcap основа	+	+	-
Визначення прихований даних	-	+	+
Використання місця на диску	484КБ	449МБ для Unix 89МБ для Windows	32МБ
Відображення в додатку шару протоколу	-	+	+
Декодування протоколу	Лише Hex, ASCII	Лише Hex, ASCII	EBDIC, Hex, ASCII
Відновлення TCP потоку	-	+(але лише форматоване)	+
Виявлення ненормальних даних	-	-(лише створює попередження)	+
Кілька інтерфейсів	-	-	+
Сповіщення знаходження	-	-	+
Відновлення HTTP веб сторінки	-	-(показує актуальний контент трафіку індивідуально)	-(показує лінки контенту трафіка індивідуально)
Мережева комунікаційна матрична мапа	-	-	+
Оцінка критичного та некритичного для бізнесу трафіку	-	+(за допомогою створення нових фільтрів та пошуку)	+(вбудована)
Можливість розробляти та налаштовувати розробниками	+(можливе налаштування користувачем під себе)	+	-(лише командою розробки capsa.co)
UDP трафік	-	+	+

FATT(Fingerprint all the things)

- На основі попереднього аналізу існуючих рішень з точки зору базового функціоналу пакетного сніфера було створено власний — FATT(Fingerprint all the things). Розроблено скрипт для вилучення метаданих та відбитків. Загальний опис створеного продукту:

★ Розроблене рішення є кросплатформним(Linux, macOS та Windows).

★ З можливістю вилучення метаданих та цифрових відбитків з різних джерел трафіку з поточного та файлу пакетних даних мережі(.pcap).

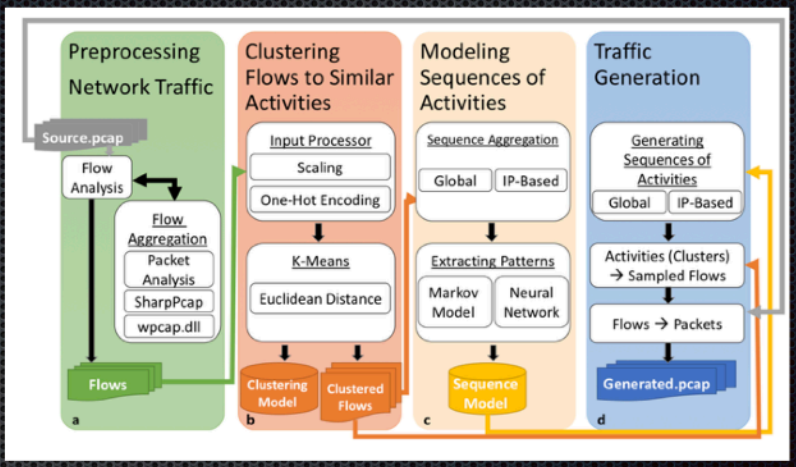
★Реалізована підтримка протоколів SSL/TLS, SSH, RDP, HTTP. Для вилучення цифрових відбитків пристроїв було обрано такі методи реалізації для відповідних протоколів:

★ *JA3 для протоколу TLS.*

★*HASSH для протоколу SSH*

★RDFP для протоколу RDP.

Фази моделювання мережевого трафіку



- вихідний трафік попередньо обробляється на агреговані потоки;
- потоки кластеризовані в групи мережевої діяльності;
- модель послідовності навчається для витягування загальних зразків на основі послідовностей мережевих дій (для цього завдання підтримується кілька методів);
- генеруються нові послідовності мережевих дій і потім перетворюються на вибіркові потоки пакетів, які відповідають кожній діяльності.

Висновки

Таким чином формуємо результати даної роботи:

1. Досліджено процес аналіза трафіку.
2. На основі цього дослідження проведено порівняльний аналіз існуючих засобів аналізу трафіку. Реалізовано базовий функіонал пакетного аналізатора для деяких мережевих протоколів з унікальною реалізацією методів створення відбитків.
3. З розробленого аналізатора з'являється потреба моделювання мережевого трафіку відповідно проведено дослідження моделювання трафіку. Базуючись на цьому розроблена та теоретично досліджена модель моделювання мережевого трафіку зі збереженням послідовності.

Перспективні напрямки розвитку

1. Виконання додаткових досліджень для створеного методу моделювання трафіку.
2. Виконати реалізацію створеного методу моделювання. Провести практичне дослідження для даного методу.
3. Проведення більш детального аналізу мережевих аналізаторів на рівні спеціалізованих бібліотек.