Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи та засоби аналізу мережного трафіку

(тема)

Виконав:
студент II курсу, групи СПзм-18-2
Сидоров Д.С.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: проф. Міхаль О.П.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Сидорову Дмитру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби аналізу мережного трафіку

затверджена наказом по університету від "30" березня 2020 р. № 43 Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2020 р.

3. Вхідні дані до роботи
аналізатор трафіку
виявлення проблем в роботі мережі
запобігання мережевих атак
діаграма класів для опису логічних з'єднань
Network Tree

4. Перелік питань, що потрібно опрацювати в роботі
Огляд предметної області
Методи та моделі представлення розбору мережевого трафіку
Розробка програмного забезпечення для аналіза трафіку

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 15 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд протоколів корпоративних	31.03.20-12.04.20	
	комп'ютерних мереж		
2	Вибір та обгрунтування методики	13.04.20-20.04.20	
	дослідження		
3	Вибір інструментальних засобів	21.04.20-27.04.20	
4	Проведення експериментів	28.04.20-05.05.20	
	Оформлення ПЗ	06.05.20-11.05.20	
6	Подання атестаційної роботи	12.05.20-13.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Міхаль О.П.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 75 с., 24 рис., 5 табл., 1 дод., 25 джерел.

КОМП’ЮТЕРНА МЕРЕЖА, ІНТЕРНЕТ, МАРШРУТИЗАТОР, ПРОТОКОЛ, СЕРВЕР, ШЛЮЗ, FIREWALL, WI-FI, WLAN, ТРАФІК.

Метою атестаційної роботи є дослідження методів та засобів аналізу мережного трафіку.

В роботі досліджені методи аналізу мережного трафіку у комп’ютерній мережі. Основне завдання реалізації управління трафіком полягає в гарантованому забезпеченні якості обслуговування доставки інформації користувачам і ефективне використання ресурсів мережі. У зв’язку з цим основне завдання реалізації управління трафіком полягає в гарантованому забезпеченні якості обслуговування доставки інформації користувачам і ефективне використання ресурсів мережі. Будь-який комутатор або маршрутизатор локальної мережі корпоративного рівня в аспекті її топологічної структури розглядається як вузол, на який надходять потоки пакетів, де вони відповідним чином обробляються і передаються на вихід. Запропоновано модифікований підхід аналізу трафіку у комп’ютерній мережі шяхом додаткового аналізу шляху, який проходить пакет по мережі, що аналізуватиме послідовность системи черг. Запропоновано метод, в якому система управління трафіком буде складатися з двох основних підсистем: системи управління установки з’єднань і системи управління комп’ютерною мережею.

ABSTRACT

Master’s thesis: 75 pages, 24 figures, 5 tables, 1 appendics, 25 sources.

FIREWALL, GATE, INTERNET, PROTOCOL, ROUTER, SERVER, WI-FI, WIRELESS NETWORK, WLAN.

The major goal of this thesis is to study the methods and means of network traffic analysis.

The methods of analysis of network traffic in a computer network are investigated in the work. The main task of traffic management is to ensure the quality of information delivery services to users and efficient use of network resources. In this regard, the main task of the implementation of traffic management is to ensure the quality of service delivery of information to users and efficient use of network resources. Any switch or router of a local area network at the corporate level in terms of its topological structure is considered as a node to which packet flows arrive, where they are appropriately processed and transmitted to the output. A modified approach to traffic analysis in a computer network is proposed by additional analysis of the path that the packet passes through the network, which will analyze the sequence of the queue system. A method is proposed in which the traffic management system will consist of two main subsystems: the connection management system and the computer network management system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

СОВ – система виявлення вторгнень

DPI – глибока перевірка пакетів

SSL – рівень захищених сокетів

TCP – протокол управління передачею

TLS – протокол захисту транспортного рівня

UDP – протокол датаграм користувача

VPN – віртуальна приватна мережа

ВСТУП

Активний розвиток інформаційних технологій зробив їх невід’ємною частиною побуту, виробництва, сфери послуг. Інформаційні системи сьогодні експлуатуються як в комерційних, так і в державних організаціях. Взаємодія між такими системами здійснюється за допомогою глобальної мережі. Спостерігається стрімке зростання обсягу мережевого трафіку, ускладнюється його структура. Аналіз трафіку стає все більш затребуваним в областях контролю і управління, оптимізації, захисту від шкідливих впливів.

Актуальними є дослідження в області зворотної інженерії закритих мережевих протоколів [1], які використовуються комерційними додатками. Тексти програм таких додатків не публікуються у відкритому доступі: розробник надає кінцевому користувачеві програму у вигляді набору виконуваних файлів. У таких випадках необхідне проведення комплексного аудиту безпеки коду на предмет пошуку помилок і вразливостей, в тому числі в реалізації використовуваного протоколу. У випадку мережевого додатку разом з бінарним кодом аналізується мережевий трафік комп’ютера (мережевого інтерфейсу), на якому цей додаток запущено.

Важливим практичним завданням є розробка методів захисту внутрішніх мереж підприємств. Один з методів полягає в накопиченні бази знань про структуру проведених атак і характерних сигнатур переданих даних. Для випадків, коли факт проведення мережевої атаки встановлений, необхідно провести розслідування: з’ясувати, як розвивався інцидент, які дані були пошкоджені або прочитані, які взаємодії з іншими комп’ютерами відбувалися. Детальний аналіз подібних інцидентів в перспективі дозволяє удосконалювати засоби захисту (як програмні, так і апаратні) таким чином, щоб блокувати певні види мережевих атак в режимі реального часу. Атаки,

засновані на перекритті IP фрагментів [2, 3] або TCPсегментов [4, 5], не можуть бути виявлені шляхом аналізу вмісту мережевих пакетів окремо. Для детектування цих атак необхідно провести IP-дефрагментацію і відновити TCP-потоки. Відновлення потоків даних, переданих між мережевими додатками, дозволяє виявляти випадки поширення шкідливого ПО, а також забороненого контенту. Відомо, що мережеві пакети можуть бути отримані приймаючою стороною в порядку, відмінному від того, в якому вони були відправлені [6]. Для відновлення потоку даних в таких випадках повинен бути відновлений порядок відправки пакетів. За статистикою частка зашифрованого трафіку істотно збільшилася за останні кілька років: великі сайти почали використовувати протокол HTTPS за замовчуванням [7, 8].

У зв’язку з цим актуальною практичним завданням є аналіз трафіку на предмет використання серверами вразливих криптографічних алгоритмів, а також недовірених сертифікатів безпеки.

Широке поширення також отримали тунельні протоколи: зокрема, вони використовуються для організації VPN [9]. Особливість тунельних протоколів полягає в тому, що потенційно, можлива побудова тунелю (тобто стека використовуваних протоколів) довільної конфігурації. Аналіз трафіку вкладених тунелів дозволить судити про безпеку такого роду сполук.

Відзначимо, що при організації мережевого тунелю в загальному випадку може використовуватися довільний стек протоколів. В умовах відсутності інформації про стеки протоколів розбір трафіку тунельних з’єднань можливий при наявності функціональності з розпізнавання протоколів, які формують стек. Спектр інструментів, що застосовуються для вирішення практичних завдань, пов’язаних з аналізом трафіку, дуже широкий - при цьому кожен використовує власний разбір трафіку і оперує над своїм внутрішнім поданням розібраних мережевих пакетів. Як правило, інструменти не використовуються окремо, а об’єднуються в

рамках деякого комплексу. Наприклад, для забезпечення безпеки внутрішньої мережі можуть використовуватися міжмережеві екрани [10], системи виявлення та запобігання вторгнень [11], системи захисту від DDoS-атак [12] та ін. Щоб додати підтримку деякого протоколу (тобто можливість його аналізувати) в організований з окремих інструментів комплекс, потрібно створити розбір пакетів цього протоколу для кожного аналізатора відповідно до його внутрішнього подання.

Обсяг необхідної роботи істотно скоротиться, якщо всі інструменти комплексу будуть спиратися на єдине внутрішнє уявлення. Завдання забезпечення безпеки мережі, а також контролю якості зв’язку вирішуються в online-режимі, тоді як розслідування інцидентів порушення інформаційної безпеки, зворотна інженерія та налагодження протоколів в offline-режимі. Під online-аналізом (або режимом) розуміється аналіз трафіку мережевого інтерфейсу (або каналу зв’язку). Offline-аналіз (або режим) передбачає аналіз попередньо збереженого в файл трафіку. Для проведення online-аналізу інструмент повинен працювати безперервно з продуктивністю, достатньою для розбору надходить трафіку: потрібне забезпечення можливості обробки потенційно нескінченного вхідного потоку даних. У разі offline-аналізу інструмент отримує вхідні дані (кінцевого розміру) з файлу. Тому може проводитися більш детальний аналіз в порівнянні з online-аналізом на аналогічному трафіку. Основна маса існуючих інструментів і застосовуваних в них методів фактично призначена для проведення аналізу тільки в одному з режимів.

У той же час, наявність універсального для двох режимів інструмента дозволило б спростити розширення функціональності. Застосування в online-аналізі розбирача, створеного в offline-режимі, фактично реалізує методологію повторного використання коду (code reuse). З плином часу розбирачі будуть доопрацьовуватися в offline-режимі і відразу ж застосовуватися для аналізу в режимі online. Таким чином, необхідна уніфікація компонентів інструментів, що відповідають за

проведення розбору мережевих пакетів. При цьому одні й ті ж розбирач повинні застосовуватися при проведенні аналізу в online і offline режимах.

Метою роботи є дослідження методів та засобів аналізу мережного трафіку.

Завданнями є:

- аналіз існуючих моделей уявлення мережевих пакетів, враховучи особливості передачі данних по мережі (втрата окремих пакетів, стиснення і шифрування даних,вкладене тунелювання);

- розробка аналізатора мережного трафіку з використанням досліджених методів.

Об’єктом дослідження є передача пакетів даних у комп’ютерній мережі.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Мережеві пакети

Аналіз мережевого трафіку набуває все більшої актуальності 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.

Тут і далі розглядаються мережі з комутацією пакетів. Рішення практичних завдань аналізу головним чином спирається на розбір заголовків мережевих протоколів в пакетах і відновлення потоків даних, що передаються. Відповідно до кількості розбираємо заголовків протоколів, що належать різним рівням моделі OSI [18], виділяють три основні класи проведеного розбору (рисунок 1.1): поверхневий (SPI - Shallow Packet Inspection), середній (MPI - Medium Packet Inspection) і поглиблений (DPI - Deep Packet Inspection) [19].

До аналізатора «поверхневого» рівня головним чином відносяться найпростіші міжмережеві екрани: рішення про блокування того чи іншого пакета зазвичай приймається відповідно до списку заборонених IP-адрес і номерів портів. Інструменти, які стосуються «середнього» рівня, дозволяють проводити фільтрацію трафіку з використанням інформації про формат переданих даних, а також більш повної (в порівнянні з окремим IP-адресою) локалізації відправника. Як правило, такі інструменти виступають в ролі

посередника (application proxy) між провайдером доступу до Інтернет і внутрішньою мережею.

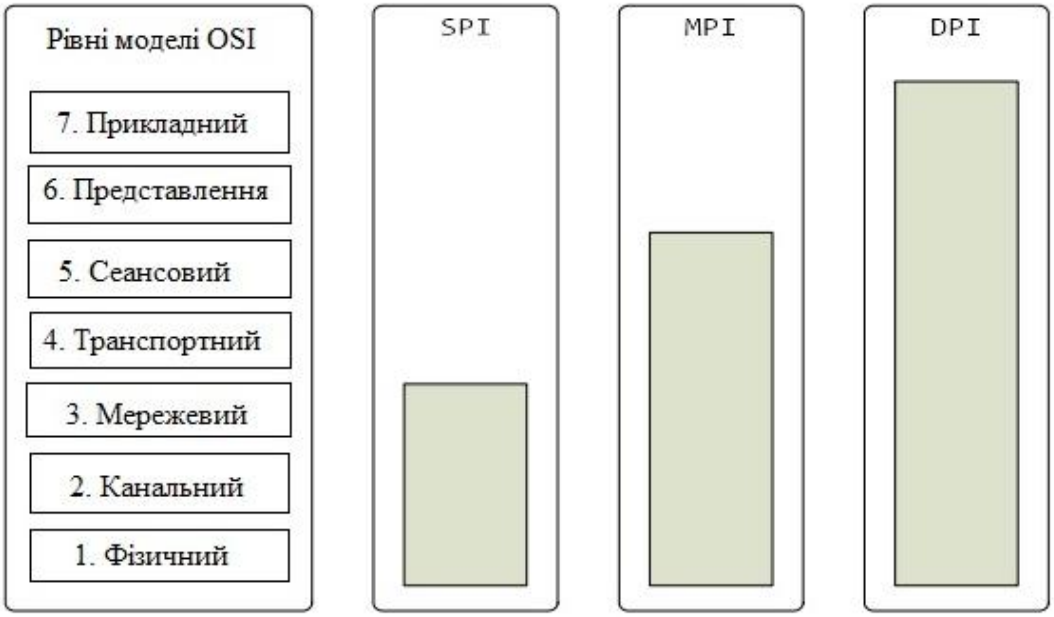

Рисунок 1.1 – Класи проведеного розбору мережевих пакетів

Системи DPI насамперед призначені для ідентифікації додатків, які беруть участь в мережевих взаємодіях. Тому «поглиблений» розбір передбачає аналіз вмісту мережевих пакетів на всіх рівнях. Для більш точної ідентифікації DPI-інструменти додатково можуть використовувати непрямі ознаки, властиві певним мережевим додаткам і / або протоколам.

1.2 Передача 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 1.2).

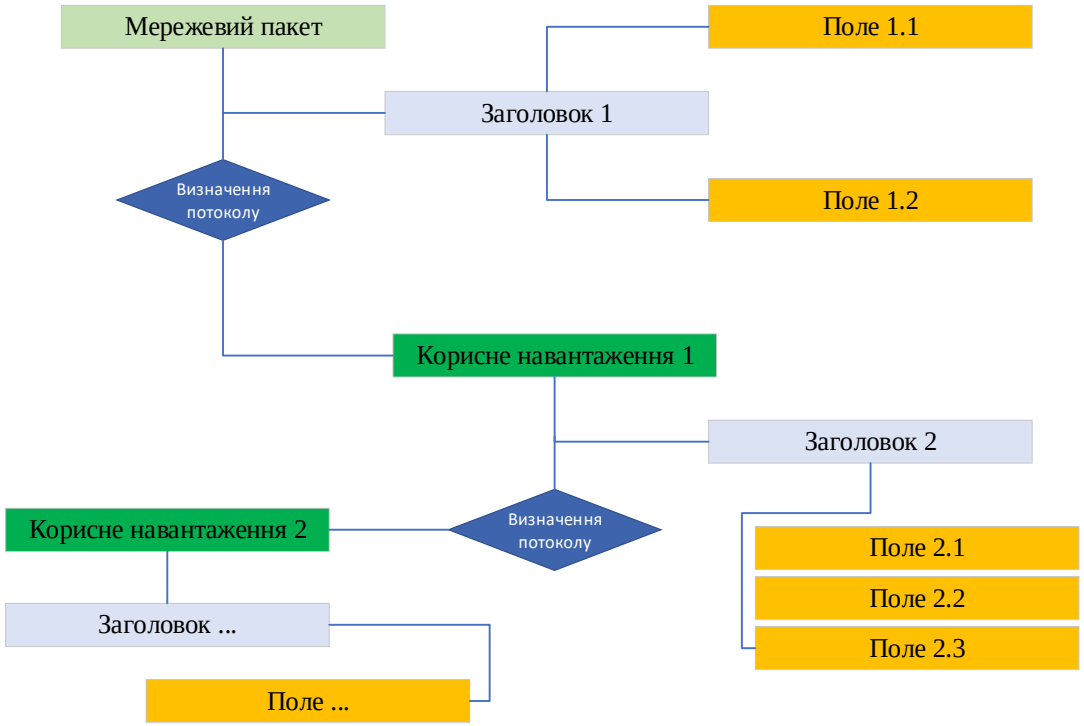

Рисунок 1.2 – Виділення і розбір заголовків протоколів в пакеті

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. 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 зростати. Тому аналізатор повинен гнучко управляти розподілом доступнихему 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 інкапсулюються 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 1.3) дані наступної PDU (Protocol Data Unit - одиниця передачі).

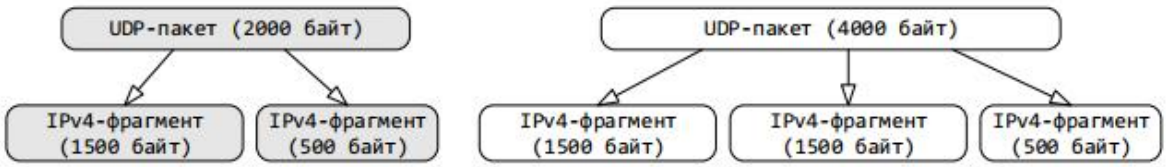

Рисунок 1.3 – Приклад фрагментації Ipv4

У разі протоколу TCP (рисунок 1.4) неформальним ознакою «останнього» для заданої PDU сегмента являє PSH-прапор, однак цей сегмент в загальному випадку містить дані наступної одиниці передачі - виникає завдання визначення меж.

Рисунок 1.4 – Приклад сегментації TCP

Для проведення глибокого аналізу, таким чином, необхідно: відновлювати вихідний порядок проходження пакетів; визначати межі верхніх PDU. Питання, пов’язані з переупорядкуванням пакетів протоколів IP і TCP, зокрема, розглядаються в статтях [22, 23]. Для забезпечення безпеки з’єднань деякі протоколи припускають передачу даних в зашифрованому вигляді (наприклад, сімейство протоколів TLS [24]). Щоб проаналізувати зашифровані дані, необхідно їх попередньо розшифрувати, використавши наданий користувачем ключ (рисунок 1.5): аналізатор повинен володіти інтерфейсом для додавання якого бракує для проведення аналізу даних.

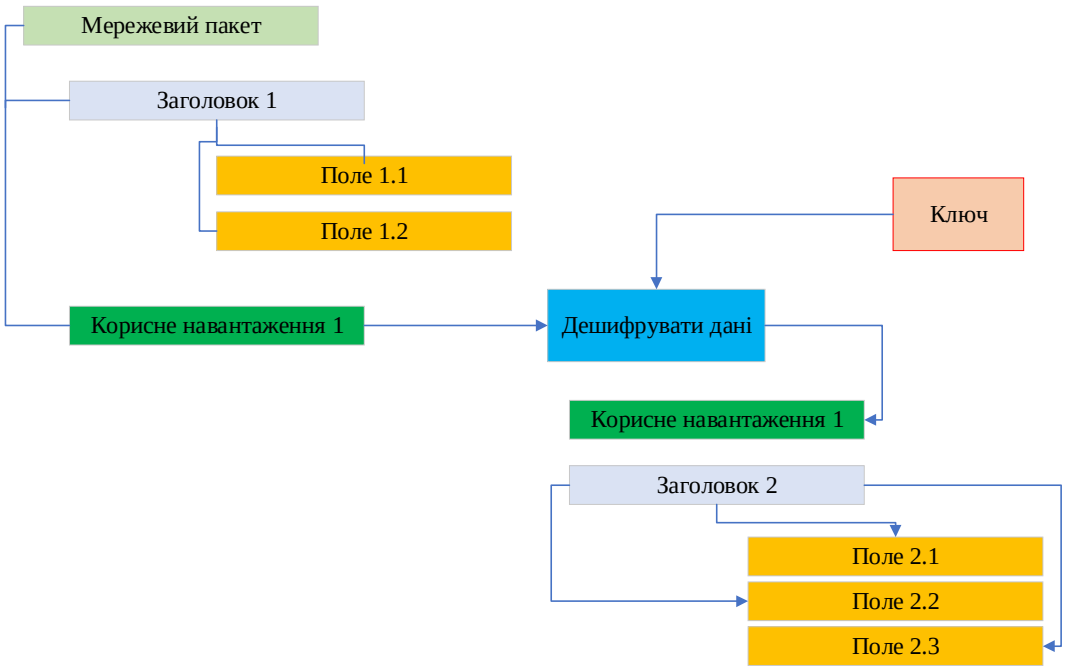

Рисунок 1.5 – Розшифровка даних з використанням ключа

При аналізі трафіку неминуче виникають помилки розбору. Під помилкою розбору розуміється невідповідність між специфікацією протоколу (кодом, що здійснює розбір) і даними, розбір яких проводиться відповідно до цієї специфікації. Причини виникнення помилок розбору різні:

- недокументовані можливості протоколу;

- спотворення даних при передачі по мережі;

- помилки в коді аналізатора. Помилки розбору повинні бути легко локалізуемое і відтворювані. Якщо виникла помилка не є критичною, аналіз повинен тривати.

1.3 Вимоги до розбору мережевого трафіку

Аналізатори мережевого трафіку, як правило, мають модульну архітектуру: 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 скрутна. До того ж потрібно повторна збірка змінених модулів. Тому необхідно мінімізувати кількість внесених в існуючі разборщікі змін, необхідних для додавання підтримки нового протоколу. Відзначимо, що деякі практичні завдання вирішуються за допомогою аналізу файлу зі збереженим трафіком (будемо називати такий файл мережевий 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

переносимість 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.

З урахуванням особливостей процесу передачі даних по мережі можно визначити функціональні вимоги до аналізаторів трафіку:

- відновлення та подальше розбір потоків даних з урахуванням можливого переупорядочивания і / або втрати окремих пакетів;
- аналіз вкладених тунелів;
- аналіз зашифрованих (модифікованих) даних;
- автоматичне визначення протоколу вищого рівня;
- локалізація та відтворення помилок розбору;
- додавання підтримки протоколів без внесення змін до ужесуществующіе модулі розбору;
- переносимість модулів розбору між режимами offline- і online-аналізу.

Вимоги 1-3 умовно можна віднести до способу представлення розібраних мережевих пакетів, вимоги 4-7 - до архітектурних особливостей підсистеми розбору і інструменту в цілому. В якості додаткової вимоги слід згадати про наявність графічного інтерфейсу.

1.4 Сучасні методи розбору

Перевірка на відповідність вимогу №1 проводиться шляхом тестування інструментів на наборі мережевих трас №1 (таблиця 1.1).

Аналіз вкладених тунелів з довільною конфігурацією стека (вимога №2) головним чином спирається на можливість автоматичного визначення протоколів (вимога №4). Тому для перевірки відповідних вимог проводиться дослідження механізмів розпізнавання протоколів в розглянутих анализаторах, а також способи організації внутрішнього уявлення розібраних

пакетів (вивчається вихідний код). Додатково на наборі мережевих трас №2 тестується підтримка відомих тунельних протоколів (таблиця 1.2).

Для інструментів, що підтримують аналіз зашифрованих (в загальному випадку модифікованих) даних, здійснюється перевірка вимоги №3 на наборі мережевих трас №3 (таблиця 1.3).

Відповідність вимогам 5-7 встановлюється безпосередньо шляхом вивчення вихідного коду (з виявленням архітектурних особливостей) і супровідної документації інструментів.

Таблиця 1.1 – Набір мережевих трас №1

Мережева траса	Особливості
Trace11.pcap	- перегрупування TCP-сегментів; - TCP-з’єднання, встановлені до початку запису траси.
Trace12.pcap	- перегрупування TCP-сегментів; - повторна передача TCP-сегментів
Trace13.pcap	- повторна передача TCP-сегментів
Trace14.pcap	- повторна передача TCP-сегментів

Таблиця 1.2 – Набір мережевих трас №2

Мережева траса	Стек протоколів
Trace21.pcap	ETH-IPv4-IPv4-ICMP
Trace22.pcap	ETH-IPv4-GRE-IPv4-ICMP
Trace23.pcap	ETH-IPv4-UDP-Teredo-IPv6-ICMPv6
Trace24.pcap	ETH-VLAN-IPv6-IPv4-GRE-PPP-IPv4-UDP-DNS

Таблиця 1.3 – Набір мережевих трас №3

Мережева траса	Особливості
Trace31.pcap	Зашифроване SSL-з’єднання
Trace32.pcap	Передача стиснутих даних за допомогою HTTP

1.5 Системи виявлення й запобігання вторгнень

Виявлення вторгнень - це процес моніторингу та подальшого аналізу подій в комп'ютерній системі або мережі з метою виявлення серед них інцидентів, які порушують встановлені політики та / або стандарти безпеки. Система виявлення вторгнень (СВВ) - це програмне або апаратне засіб, що автоматизує виявлення вторгнень. За способом виявлення системи поділяють на сигнатурні (signature-based) і аномальні (anomaly-based). Перші виявляють мережеві атаки за допомогою пошуку попередньо підготовлених шаблонів (патернів) в трафіку, що надходить в підмережа або на окремий комп'ютер. Сигнатурні системи, таким чином, здатні виявляти лише відомі види атак, тоді як і зустрічалися раніше атакам вони нічого не можуть протиставити. Аномальні системи навпроти орієнтовані на виявлення нових атак: загальна ідея підходу полягає в застосуванні методів машинного навчання для побудови моделі безпечної поведінки та подальшого порівняння цієї моделі з піднаглядним поведінкою. Аномальні системи зазвичай характеризуються великою кількістю хибнопозитивних спрацьовувань.

Snort. Робота над Snort почалася в 1998 році. Інструмент являє собою сигнатурну СОВ і призначений для запобігання мережевих атак. Основні компоненти системи - це набір препроцесорів, призначених для розбору мережевих пакетів, і модулі виявлення, реалізовані за допомогою завдання правил виду Action-Connections [-Options]. Правила застосовуються до мережевих пакетів і діляться на статичні (виконуються завжди) і динамічні (виконуються в разі спрацювання будь-якого іншого правила). Action визначає дію, яке виконує в разі спрацювання правила: зберегти або ігнорувати мережевий пакет, активувати динамічне правило і ін. Connections задає безліч TCP або UDP з'єднань, до яких слід застосовувати дане правило. В Options можуть бути визначені сигнатура і область даних мережевого пакету для її пошуку, текстове повідомлення для запису в логфайлів, ідентифікатор динамічного правила, який активується даними. Наприклад,

правило на рис. 6 полягає в наступному: якщо в корисне навантаження TCP-пакета, що надійшов на TCP-порт 111 машини з підмережі 192.168.1.0/24, буде виявлена сукупність електронних даних «00 01 86 a5», буде згенеровано попередження з повідомленням «mountd access».

The Bro Network Security Monitor. Робота над інструментом Bro почалася в 1995 році. На даний момент система дозволяє аналізувати мережевий трафік на предмет виявлення вторгнень, з метою збору і аналізу статистики, для вимірювання пропускної здатності мережі і детектування проблем в її роботі. Інструмент являє собою гібридну СВВ: передбачені інтерфейси як для сигнатурного пошуку, так і для застосування алгоритмів машинного навчання. В системі умовно можна виділити два головних компонента (рисунок 1.6): генератор подій (ядро системи) і інтерпретатор скриптів, формально описують політику безпеки (які дії слід робити в разі генерації тієї чи іншої події). Подія може виникнути в результаті розбору одного або декількох мережевих пакетів, а також внаслідок обробки іншої події (підтримуються ланцюжка подій).

Рисунок 1.6 – Установка Bro в Ubuntu

Wireshark. Робота над інструментом почалася в 1998 році. Wireshark надає функціональність, схожу з можливостями програми tcpdump з налагодження мережевих додатків і конфігурації мережі в цілому, однак має графічний інтерфейс і дозволяє виконувати фільтрацію мережевих пакетів під час захоплення або відображенні по великому числу критеріїв. На відміну від Snort і Bro Wireshark зберігає пакети в файл, після чого проводить їх розбір. Це дозволяє переглядати дані всіх полів всіх виділених заголовків у всіх розібраних пакетах. На сьогоднішній день здійснюється підтримка близько 2000 протоколів; при цьому кількість полів, які можна використовувати для фільтрації, становить близько 206000. Wireshark є найбільш популярним аналізатором трафіку серед вільно розповсюджованих інструментів.

1.6 Результати оцінки

Будемо називати мережевий протокол низькорівневим, якщо він відноситься до фізичного, канального, мережевого або транспортному рівню моделі OSI. Всі інші протоколи (сеансовий, представницький, прикладної рівні) будемо називати високорівневими.

На рисунку 1.7 наведені TCP-потоки, при відновленні яких між інструментами виникли розбіжності: знак «+» означає, що потік даних відновлений вірно, «-» говорить про наявність неточностей.

Вміст відновлених потоків іноді різниться. Аналіз результатів показав наступне:

- інструменти починають відновлювати дані TCP-з’єднань з моменту виявлення першого сегмента;

- кожне з TCP-з’єднань 1-3, 5-7 і 9 має по кілька сегментів, які потрапили в трасу в неправильному порядку. Bro додав дані цих сегментів правильно у всіх з’єднаннях. Snort додав дані в тому порядку, в якому вони

прийшли. Wireshark відновив правильний порядок сегментів для з’єднань 3, 6, 7, і не зробив цього для інших.

##	SRC (IP:PORT)	DST (IP:PORT)	Коректне відновлення потоку		
			Wireshark v.2.2.3	Snort v.2.9.7.0	Bro v.2.5
Trace11.pcap					
1	68.142.205.139:80	192.168.0.105:25168	-	-	+
2	68.142.205.139:80	192.168.0.105:25175	-	-	+
3	68.142.205.139:80	192.168.0.105:25180	+	-	+
Trace12.pcap					
4	74.125.19.113:443	192.168.0.105:24044	+	-	+
5	68.142.205.139:80	192.168.0.105:24053	-	-	+
6	68.142.205.139:80	192.168.0.105:24060	+	-	+
7	68.142.205.139:80	192.168.0.105:24089	+	-	+
Trace13.pcap					
8	74.125.224.96:443	24.6.173.220:61960	+	-	+
Trace14.pcap					
9	174.46.74.87:80	192.168.2.7:43542	-	-	+

Рисунок 1.7 – Установка Bro в Ubuntu

Це сталося через те, що в сполуках 3, 6, 7 розмір відновлюваної PDU вдалося визначити до того, як був розібраний перший сегмент, який потрапив в трасу в неправильному порядку: для цього, зокрема, використовувалося значення поля Content-Length заголовка HTTP. У з’єднаннях 1, 2, 5, 9 застосовується порційна передача «chunked», в результаті чого розмір PDU визначається тільки при отриманні останньої порції даних. Значить, при наявності хоча б однієї інверсії в порядку проходження сегментів відновлення PDU неможливо. TCP-з’єднання 4 і 8 мають сегмент повторної передачі. Дані цих сегментів були вірно відкинуті Wireshark і Bro, але помилково додані Snort.

1.6.1 Аналіз вкладених тунелів

В процесі тунелювання беруть участь наступні типи протоколів:

- транспортуємий - протокол мереж, що об’єднуються;

- протокол транзитної мережі;

- інкапсулючий протокол - поміщає пакети транспортується протоколу в поле даних пакетів несе протоколу.

При аналізі вкладених тунелів критичними для системи є можливість автоматичного визначення протоколу вищого рівня (вимога №4), а також підтримка стека протоколів довільної глибини. У Wireshark передбачено три способи для активації розбирача заголовка протоколу вищого рівня:

а) прямий виклик розбирача: безпосередній виклик заданої функції розбору в коді.

б) зворотний виклик розбирача з прив’язкою за значенням: викликається розбирач, що визначається значенням, отриманим при розборі заголовка нижчого протоколу. Так, наприклад, використовуються поля EtherType і Protocol в заголовках Ethernet і IP відповідно.

в) виклик розбирача за допомогою евристичної прив’язки. Евристики застосовуються, коли в заголовку нижчого протоколу відсутні ознаки для точного визначення наступного протоколу. Зокрема, номери портів TCP і UDP не ставлять протокол вищого рівня точно: дані, що передаються по порту TCP 80 як правило представляють собою HTTP-трафік, проте в загальному випадку це може бути не так.

Розібраний мережевий пакет в Wireshark представляється у вигляді дерева, вершини якого описують заголовки протоколів і виділені в них поля. Такий підхід дозволяє аналізувати вкладені тунельні протоколи: можливість додавання в дерево нового рівня була передбачена розробником. Більшість препроцесорів Snort жорстко пов’язані з номерами портів TCP / UDP: відповідність встановлюється за допомогою конфігураційного файлу. Таким чином, мережеві пакети протоколу P, що надходять на порт X, будуть

розбиратися тільки в разі, якщо X входить в безліч відслідковуються портів декодера P. Тільки препроцесори для SSH, DCE / RPC і SMB дозволяють автоматично визначати відповідний протокол. Ймовірно, розробник відмовився від динамічного розпізнавання протоколів в большості препроцесорів для збільшення швидкості аналізу. Мережевий пакет в Snort описується за допомогою структури з фіксуванням набором полів. Зокрема, зафіксовано максимальну кількість IP заголовків в пакеті, що дорівнює двом. При цьому, значення полів структури не можуть бути перезаписані в процесі аналізу. Тому, якщо в трафіку зустрінеться мережевий пакет з трьома IP-заголовками, то його розбір припиниться на третьому IP заголовку. В аналізаторі Bro фактично представлені ті ж механізми активації розбирачів, що і в Wireshark: передбачена прив’язка номерами портів TCP / UDP, а також до попередньо заданих сигнатурам (аналог евристичних розбирачів). На рисунку 1.8 наведені сигнатури для виявлення HTTP-з’єднань в трафіку. На відміну від Wireshark, де евристики кодуються безпосередньо на мові Ci, Bro надає мову опису сигнатур, що істотно спрощує розробку. Разбирачі протоколів канального, мережевого і транспортного рівнів викликаються безпосередньо.

```
signature dpd_http_client {
    ip-proto == tcp
    payload /^ *(GET|HEAD|POST) */
    tcp-state originator
  }

signature dpd_http_server {
    ip-proto == tcp
    payload /^HTTP\/[0-9]/
    tcp-state responder
    requires-reverse-signature dpd_http_client
    enable "http"
  }
```

Рисунок 1.8 – Сигнатури для виявлення HTTP-з’єднань в трафіку

Структура для опису мережевого пакету в Bro містить відомості тільки про канальному рівні. Інформація про мережевому і транспортному рівнях зберігається в структурі, яка описує з’єднання. Кожне таке з’єднання визначається п’ятіркою: два IP-адреси, два порти і протокол транспортного рівня. Для цього з’єднання зберігається стек тунельних з’єднань, в рамках яких воно було організовано. Кожен елемент такого стека є або пару IP-адресу (для опису інкапсуляції виду IP-in-IP), або п’ятірку, яка описує з’єднання несе протоколу. Передбачено визначення максимально можливої вкладеності тунелю: якщо кількість елементів в стеку тунельних з’єднань перевищить граничне значення, аналіз верхніх заголовків проводитися не буде. Тестування показало, що на всіх трасах набору №2 Wireshark і Bro вірно визначили і розібрали всі вкладені заголовки протоколів. У Snort виникли труднощі при аналізі Trace24.pcap: заголовки ETH-VLAN-IPv6-IPv4-GRE були розібрані, заголовки PPP-IPv4-UDP-DNS - немає. Такий результат Snort повністю узгоджується з документацією.

1.6.2 Аналіз модифікованих даних

Wireshark підтримує можливість аналізу модифікованих даних: для розбору зашифрованих пакетів в рамках заданого протоколу слід вказати шлях до файлу з ключами для дешифрування, розпакування стислих даних здійснюється автоматично. Snort не припускав проведення дешифрування з міркувань продуктивності. Для препроцесорів протоколів, що підтримують шифрування, передбачена спеціальна опція «ігнорувати зашифрований трафік» з тим, щоб зменшити число хибнопозитивних спрацьовувань. У Bro теж не передбачається можливість розшифровки. Декомпресія даних підтримується як в Snort, так і в Bro.

1.6.3 Локалізація помилок розбору

Wireshark дозволяє реєструвати помилки, що виникають в результаті невідповідності розглядаємого пакета формату, визначеного розбирачем відповідного протоколу: в графічному інтерфейсі передбачена компонента відображення всіх таких помилок, яка також дозволяє перемикатися до «потрібного» мережевого пакету. Snort і Bro не передбачають збереження всього проаналізованого трафіку, однак за допомогою правил і подій може бути реалізована запис в файл пакетів, розбір яких призводить до виникнення помилки. Надалі потрібно використовувати програму перегляду збережених пакетів (наприклад, hex-editor) для того, щоб з'ясувати які саме байти в них не відповідають специфікації, що важко.

Wireshark, Snort і Bro дозволяють аналізувати трафік, що надходить на мережевий інтерфейс, а також попередньо збережений у файлі. При цьому використовуються одні й ті ж разборщікі заголовків.

Підтримка нового протоколу без модифікації існуючого коду в Wireshark може бути реалізована за допомогою прив'язки за значенням або евристичної прив'язки. Модулі розбору протоколів всіх рівнів в Wireshark реалізовані одноманітно, чого не можна сказати про Snort і Bro: якщо разборщікі протоколів канального, мережевого і транспортного рівнів жорстко зв'язані за допомогою прямих викликів, то аналіз високорівневих протоколів виконується з урахуванням можливостей динамічного розпізнавання (Bro) або завдання діапазонів портів (Snort). Таким чином, для Bro і Snort підтримка нового низькорівневого протоколу зажадає змінити існуючий код. Модулі розбору в Wireshark не є незалежними: SSL-разборщик використовує інформацію про відновлення PDU з декількох TCP сегментів, HTTP-розбирачів потрібні функції, що застосовуються при аналізі SSL-трафіку і ін. Подібна організація коду збільшує його зв'язність і ускладнює подальшу розробку: в разі зміни коду розбирача даного протоколу необхідно проконтролювати коректність коду у всіх залежних розбирачів.

Високорівневі декодери Snort і Bro також залежать від низькорівневих, але в значно меншому ступені, ніж в Wireshark.

1.7 Висновки до розділу 1.7

Розглянуті інструменти в повній мірі не відповідають висунутим вимогам. Модулі розбору в них сильно пов’язані на рівні вихідного коду, що передбачає проведення великого обсягу роботи в разі внесення в код змін. Тільки Bro правильно відновлює потоки даних, Snort і Wireshark не справляються з цим завданням. Незважаючи на можливість застосування одних і тих же розбирачів при проведенні online і offline аналізу, кожен інструмент завжди використовує одну і ту ж стратегію обробки пакетів. Snort і Bro видаляють (drop) пакет відразу після проведення розбору, що утруднює їх застосування для налагодження розбирачів (offline). При проведенні online-аналізу Wireshark зберігає всі мережеві пакети в фізичної пам’яті, що, з огляду на кінцівки обсягу пам’яті, не дозволяє проводити розбір протягом тривалого часу. Для досягнення поставленої мети необхідно розробити модель представлення даних, а також систему аналізу трафіку на її основі.

2 МЕТОДИ ТА МОДЕЛІ ПРЕДСТАВЛЕННЯ РОЗБОРУ МЕРЕЖЕВОГО ТРАФІКУ

Будемо розглядати мережеві взаємодії між додатками, що не фіксують будь-який конкретний стек протоколів (TCP / IP, IPX / SPX). Будемо вважати, що мережевий обмін здійснюється за допомогою довільного стека протоколів, що нараховує N рівнів (рисунок 2.1). Мережева взаємодія розглядається як упорядкований набір логічних з'єднань. В рамках кожного з'єднання передається послідовність пакетів між логічними об'єктами одного рівня.

Рисунок 2.1 – Мережева взаємодія між двама додатками

Нехай логічне з'єднання на і-му рівні (і = 1…N) реалізується за допомогою протоколу містить пакетів. Кожен протокол $Protocol^i$ задає безліч допустимих форматів пакетів $\{F_j\}$. Під форматом будемо розуміти поділ послідовності байт пакету на іменовані поля і обмеження на значення деяких полів. Як правило, корисне навантаження - дані протоколу верхнього рівня - поміщена в одне поле, інші поля виконують службові функції: вказують тип формату повідомлення, несуть в собі адресну інформацію, задають розміри інших полів і т.п. Спрощено пакет протоколу $Protocol^i$ розглядатимемо як

кортеж, що складається із сукупності полів з керуючою інформацією і корисного навантаження:

$$Packet^i = <Control^i, Payload^i> , i = 1 \dots N$$

Всі мережеві протоколи можна розділити на два непересічних безлічі: допускають і не допускають фрагментації даних. Фрагментація, в сво чергу, може бути організована різними способами.

При спостереженні стану підключення до мережі фіксується проходження по ньому послідовності пакетів протоколу нижнього рівня. Пакети відносяться до довільному числу додатків, попарно взаємодіючих або виконують трансляцію розсилку даних. Повний відтворення функціональності мережевого протоколу при розборі пакетів не потрібно, необхідно розділяти потік байт на окремі пакети (повідомлення), вибираючи необхідний формат, а потім інтерпретувати керуючу інформацію Controli з метою угруповання пакетів і вилучення корисного навантаження.

Поглиблений аналіз трафіку на увазі відновлення і угруповання послідовності пакетів для всіх логічних з’єднань всіх рівнів.

Порядок отримання пакетів може відрізнятися від порядку їх відправлення, тому для протоколів, що допускають фрагментацію, на основі інформації, що управляє пакету повинні будуватися предикати передування (визначення попереднього фрагмента) і проходження (визначення наступного фрагмента), за допомогою яких буде проводитися дефрагментація. При переході до аналізу пакетів вищого логічного з’єднання пакети даного логічного з’єднання повинні бути переупорядкувати відповідно до значень введених предикатів.

2.1 Сутності моделі

Далі описується модель уявлення розібраних мережевих пакетів, що задовольняє перерахованим в підрозділі 1.2 вимогам і дозволяє розглядати мережеві взаємодії у вигляді послідовності логічних з’єднань.

2.1.1 Мережевий пакет

На рисунку 2.2 зображена діаграма класів, за допомогою яких представляється розібраний мережевий пакет. Для відновлення послідовності пакетів логічного з’єднання потрібно витяг корисного навантаження пакетів нижчого логічного з’єднання. Витяг даних в розробленої моделі супроводжується їх копіюванням тільки при необхідності проведення дефрагментації. Завдяки такій стратегії підвищується продуктивність в порівнянні з підходами, які вимагають копіювати корисне навантаження при добуванні в загальному випадку.

Виділені в процесі розбору структурні одиниці (мережеві пакети і їх поля) описуються за допомогою блоків (клас Block). Блок представляє масив байт деякого фіксованого розміру. При аналізі блоку може знадобитися окремо обробити будь-яку його частину. Для цього буде створено новий блок, який є дочірнім по відношенню до вихідного батьківського блоку. У процесі аналізу кожного блоку ставлять у відповідність тип розбору (поле ParseType класу Block), що визначає функцію (поле Parser класу Type) для розбору даних цього блоку.

Розбір полягає в поділі буфера на іменовані поля відповідно до заданого формату пакета F. Для блоку визначається базовий формат даних (поле StructType класу Block). Основні базові формати: Struct - блок містить різнорідні поля, аналог типу структ мови C (ProtoStruct для випадків, коли описується заголовок мережевого протоколу); FieldSeq - послідовність однорідних полів, аналог типу масив мови C; PacketSeq - послідовність

мережевих пакетів (можливо різних форматів) одного протоколу. Вводиться два види блоків: блок-потік - для опису відновлених потоків даних, і блок-фрагмент - для представлення цілих пакетів і окремих полів, виділених в потоках даних.

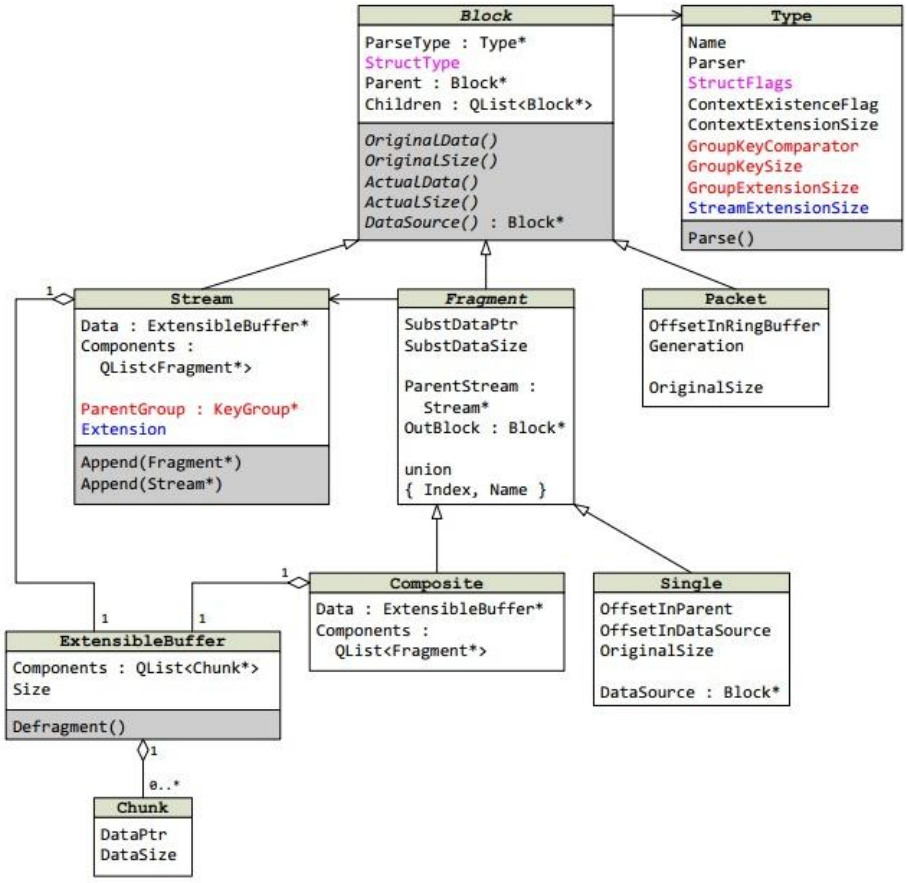

Рисунок 2.2 – Діаграма класів для опису розібраних мережевих пакетів

Відновлення потоку має на увазі об’єднання полів (як правило, це поле з корисним навантаженням) з різних мережевих пакетів. З цією метою вводиться поняття накопичувального буфера (клас ExtensibleBuffer), що дозволяє об’єднувати як дані блоків фрагментів, так і дані блоків-потоків. Відзначимо, що для блоку-потоку блок-батько не визначений, оскільки відношення «батько-нащадок» не застосовується для відновлених потоків.

Для розміщення в пам’яті розшифрованих (або розпакованих) даних вводиться поняття заміщує буфера: його розмір фіксований, на відміну від

накопичувального буфера, що збільшується в міру додавання даних. Блокфрагмент на непустому заміщує буфером будемо називати заміщує блоком-фрагментом: подальшого розбору підлягають дані заміщає буфера. Дочірні блоки заміщає блоку-фрагмента характеризуються зміщенням щодо відповідного заміщає буфера. Відзначимо, що розмір заміщає буфера ніяк не пов’язаний з розміром вихідних (зашифрованих або стислих) даних блоку-фрагмента. За визначенням, блокпоток не може містити заміщає буфер. Замісник блок-фрагмент, таким чином, характеризується початковими (зашифрованими) і фактичними (розшифрованими) даними. Узагальнюючи ці поняття для блоку, будемо називати фактичними дані блоку, підлягають подальшому розбору, і початковими дані без урахування можливого заміщення. Для блоку-потоку початкові дані збігаються з фактичними. Вводиться два види блоків-фрагментів: складовою (клас Composite) - для опису дефрагментірован PDU, і одиночний (клас Single) - для подання простих полів. Для стислості надалі будемо називати складовою блок-фрагмент складовим фрагментом, а одиночний блок-фрагмент - одиночним фрагментом. Дефрагментація PDU, як і відновлення потоку, має на увазі об’єднання полів з різних пакетів, тому складовою фрагмент володіє розширюваним буфером (поле Data класу Composite). У той же час, складовою фрагмент призначений для опису однієї PDU, тоді як в буфері блоку-потоку, як правило, міститься кілька PDU. Будемо називати джерелом даних блоку (поле DataSource класу Block) блок, що містить буфер, в якому містяться початкові дані першого. Значить, в якості джерела даних може виступати блок потік, складовою фрагмент, який замінює блок-фрагмент. Одиночний блок локалізується в даних свого джерела за допомогою зміщення (поле OffsetInDataSource класу Signle). Відповідно до структурним типом батьківського блоку блок-фрагмент описує або поле структури (характеризується ім’ям Name), або елемент послідовності (характеризується індексом Index).

2.1.2 Логічне з’єднання

На рисунку 2.3 представлена діаграма класів, за допомогою яких описуються логічні з’єднання.

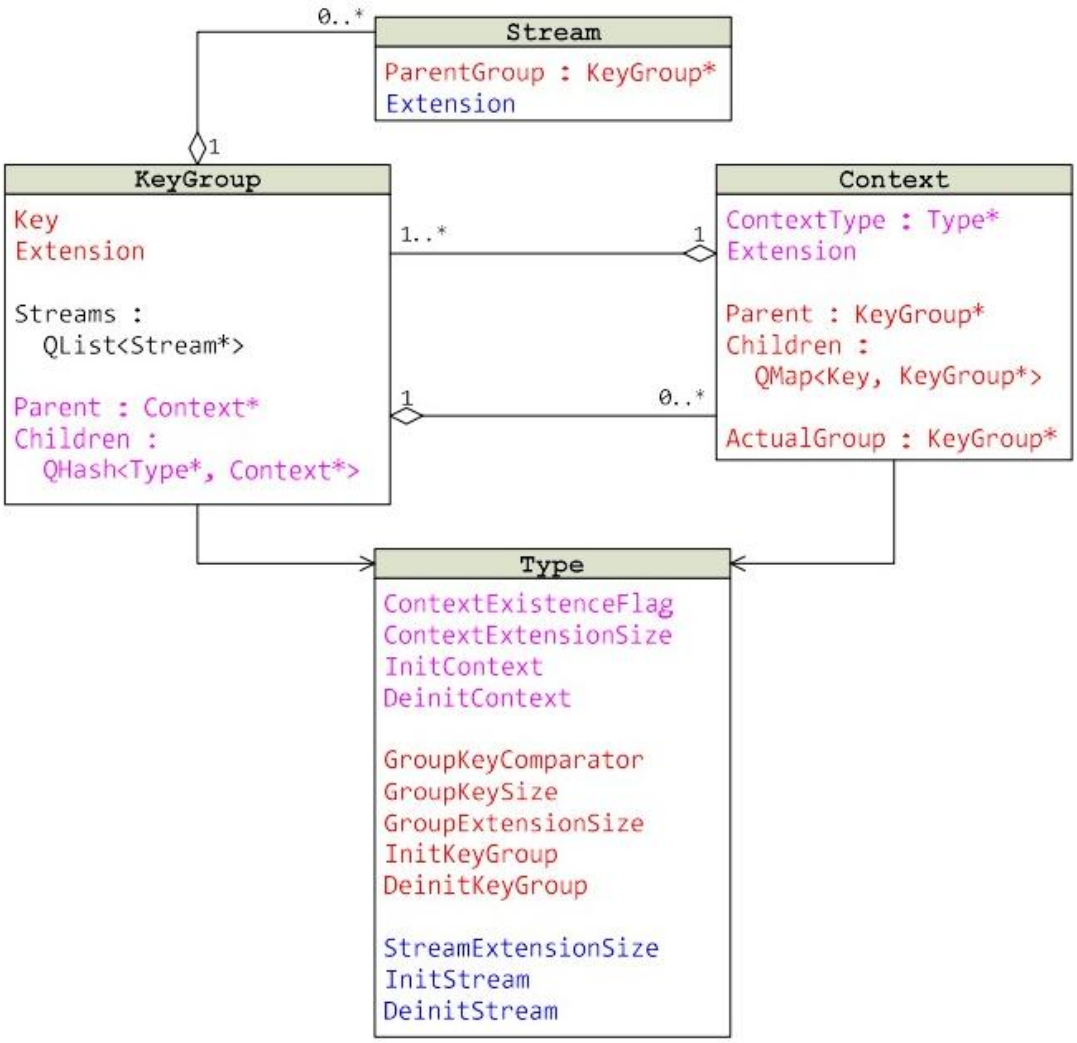

Рисунок 2.3 – Діаграма класів для опису розібраних мережевих пакетів

Кожне логічне з’єднання організовано за допомогою одного мережевого протоколу і представляється у вигляді послідовності пакетів цього протоколу. У той же час, на одному рівні стека протоколів можуть бути представлені пакети різних протоколів. Для виділення послідовностей пакетів, що відносяться до свідомо різним логічним з’єднанням, необхідно

згрупувати пакети заданого рівня стека відповідно до протоколів цих логічних з’єднань.

Група пакетів деякого протоколу в рамках заданого рівня стека описується контекстом (клас Context). Мережевий протокол, на підставі якого був сформований контекст, за визначенням задає тип контексту (поле ContextType класу Context). У розширенні (поле Extension класу Context) контексту може зберігатися додаткова інформація, необхідна для проведення розбору пакетів, що відносяться до даного контексту (наприклад, стан декомпресора при аналізі пакетів протоколу PPP). Тип контексту використовується, зокрема, при побудові ідентифікаторів логічних з’єднань. Логічне з’єднання протоколу деякого рівня повинно бути представлено послідовністю відновлених пакетів. Для відновлення (вилучення корисного навантаження) необхідно попередньо виділити відповідну підпослідовність з послідовності пакетів цього протоколу. Для угруповання пакетів в підпослідовності використовується адресна інформація з службових полів. Кожна підпослідовність описується ключовий групою (клас KeyGroup). Така група в контексті ідентифікується за допомогою унікального ключа (Key), вміст і розмір якого визначаються відповідно до типу контексту. Контекст виступає в ролі контейнера по відношенню до таких груп і надає до них доступ по ключу. Ключова група володіє розширенням (Extension) для збереження інформації, необхідної під час проведення розбору блоків (пакетів) цієї групи. Ключова група фактично являє собою логічне з’єднання. Будемо називати відновленим потоком протоколу P блок потік, в накопичувальний буфер якого додані дані блоковфрагментов однієї ключової групи (можливо переупорядкування і / або перекриття блоків фрагментів, відповідно до семантики протоколу P) контексту з типом, що описує протокол P. Відповідна ключова група називається батьківської (поле ParentGroup класу Stream) по відношенню до відновленого потоку. В рамках однієї ключової групи підтримується одночасне існування кількох потоків (поле Streams класу KeyGroup) - необхідність в цьому може бути обумовлена

як семантикою відповідного протоколу, так і втратою окремих пакетів при передачі.

Розширення потоку (поле Extension класу Stream) використовується для збереження необхідної при відновленні цього потоку інформації. Пояснимо на прикладі протоколу TCP, як керуюча інформація використовується для переупорядочивания пакетів логічного з'єднання. Початок потоку визначається TCP-пакетом з встановленим прапором SYN, кінець - TCP-пакетом з встановленим прапором FIN або прапором RST.

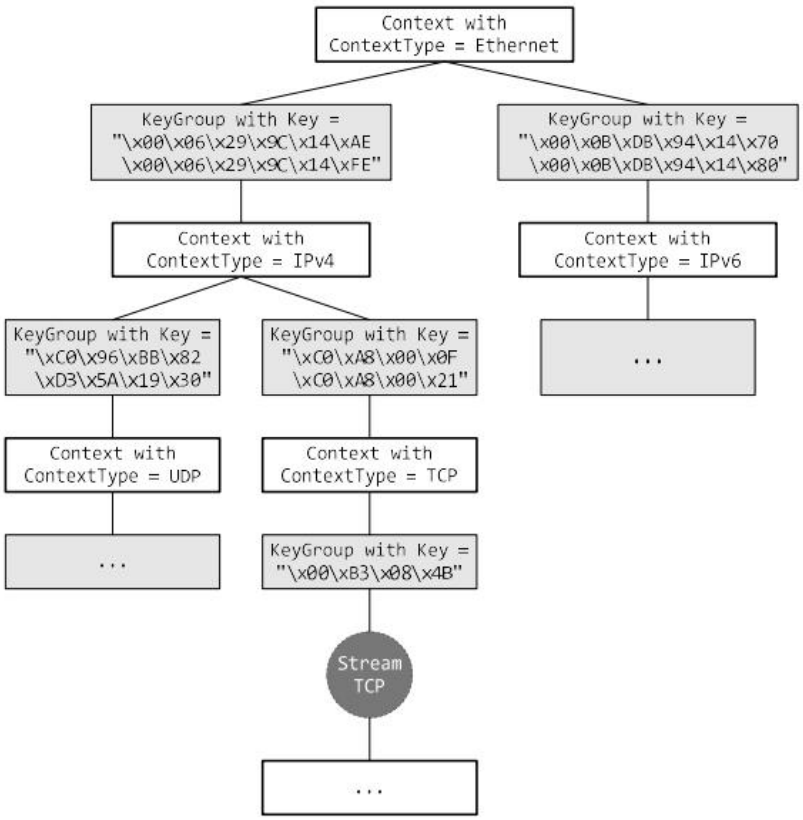

Рисунок 2.4 – Приклад дерева контекстів для мережі Ethernet

У відповідності зі значеннями порядкових номерів (SEQ) і номерів підтвердження (ACK) може бути проведено переупорядкування TCP сегментів. В результаті відновлення логічних з'єднань різних взаємодіючих по мережі додатків буде побудовано дерево контекстів, ключових груп і відновлених потоків (далі, дерево контекстів) з вершинами трьох типів:

- вершина-контекст задає протокол;

- вершина-група характеризується ключом і описує логічне з’єднання на рівні протоколу батьківської вершини;

- вершина-потік описує відновлений потік даних протоколу вищого рівня.

На рисунку 2.4 наведено приклад дерева контекстів, в якому в якості протоколу нижнього рівня, що спостерігається під час встановлення зв’язку, фігурує Ethernet. Ключ для Ethernet-групи будується шляхом конкатенації MAC-адрес відправника і одержувача (розмір ключа дорівнює 12 байтам). Для протоколу IPv4 ключ групи складається з пари IP-адрес (8 байт), для протоколу TCP - з пари портів (4 байта).

2.1.3 Учасники мережевого обміну

Ключі груп пакетів, що виділяються, є масиви байт. Це, з одного боку, дозволяє конструювати ключ і групувати пакети довільним чином, з іншого - призводить до втрати інформації про напрямок руху пакета в мережі. У той же час знання про адреси відправника і одержувача можуть використовуватися для побудови топології мережі, трафік якої аналізується, виявлення запущених на комп’ютерах служб, візуалізації мережевих взаємодій. Для вирішення цих завдань вводиться поняття мережевого вузла (клас NetworkNode), що відноситься до протоколу деякого рівня.

Для ідентифікації мережевого вузла задається відповідний протокол і імена полів в його повідомленнях, в яких міститься адресна інформація. Для уніфікації опису використовується трійка (ProtoType, ValueType, Value), де прототипом - протокол, ValueType - тип розбору блоку, що описує «адресне» поле, Value - значення «адресного» поля.

Якщо для одних протоколів дані про мережеві вузлах містяться в заголовку безпосередньо у вигляді полів, то для інших є лише непрямі ознаки: незважаючи на те, що в HTTP-заголовку в явному вигляді відсутня

поле, яке ідентифікує відправника (клієнт або сервер), можна зробити відповідний висновок, проаналізувавши вміст стартовою рядка. Для роботи з такими сценаріями Value може приймати зумовлені значення, відповідні мережевий ролі вузла: client, server, peer, unknown. При цьому значення поля ValueType буде порожнім. Адресної інформації поточного протоколу недостатньо для ідентифікації логічного з'єднання. Наприклад, при організації тунельного з'єднання GRE пакети, відправлені різними учасниками мережевого обміну, будуть містити заголовки з одними і тими ж IP-адресами. Для подолання цієї проблеми вводяться ланцюжка мережевих вузлів. Ланцюжок будується шляхом додавання в її кінець мережевих вузлів (або відправників, або одержувачів) протоколів нижчих рівнів, що виникли в процесі розбору мережевого пакету. Мережевий вузол в ланцюжку, що безпосередньо передує даному, будемо називати батьківським. Мережевий вузол, безпосередньо наступний за цим вузлом в ланцюжку, будемо називати дочірнім. Для першого вузла ланцюжка, таким чином, не визначений батьківський вузол, а для останнього - дочірній. Будемо називати ланцюжком протоколів послідовність значень поля ProtoType для заданої ланцюжка мережевих вузлів. Для компактного представлення всіх побудованих ланцюжків і організації швидкого доступу до них використовується префіксне дерево. Для гарантованої наявності загального префікса у всіх ланцюжків додамо в початок кожного ланцюжка додатковий фіктивний мережевий вузол з невизначеними ProtoType і ValueType і порожнім Value. У побудованому префіксному дереві (клас NetworkTree) шлях від кореня до листової вершині однозначно ідентифікує учасника мережевого обміну. Для опису обміну пакетами між мережевими вузлами одного рівня NetworkTree доповнюється горизонтальними ребрами (клас Edge). Горизонтальне ребро з'єднує два мережевих вузла, якщо:

- ланцюжки протоколів мережевих вузлів однакові;

- між вузлами переданий хоча б один мережевий пакет протоколу, описуваного за допомогою ProtoType. На рисунку 2.5 наведено приклад

NetworkTree, де в якості протоколів найбільш низького і високого рівнів фігурують Ethernet і HTTP відповідно.

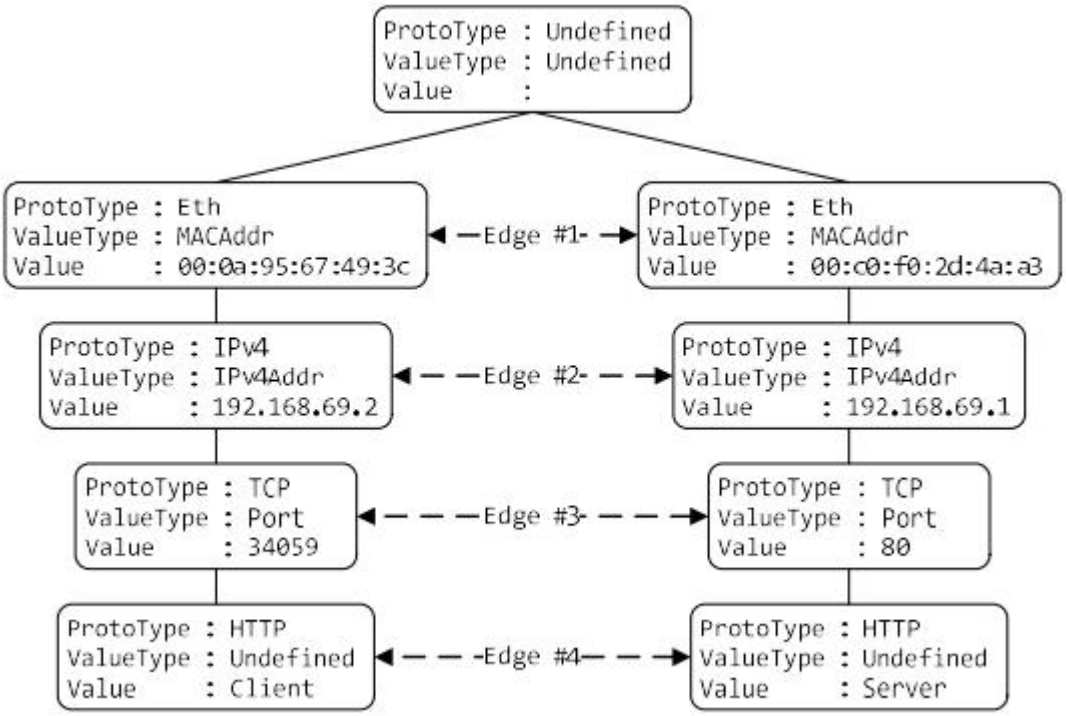

Рисунок 2.5 – Приклад NetworkTree

Для опису безлічі пакетів, переданих між парою мережеві вузлів, вводиться поняття діалогу (Dialog). Діалог описує взаємодію між вузлами з однаковими ланцюжками протоколів. Тип діалогу визначається значенням ProtoType взаємодіючих вузлів. Для кожного пакету діалогу визначено напрямок передачі: від першого вузла до другого або навпаки. Між діалогами і ребрами, таким чином, встановлюється взаємно однозначна відповідність. Для широкомовних розсилок вводяться фіктивні мережеві вузли.

NetworkTree дозволяє розглядати мережевий обмін на різних рівнях, починаючи від канального і закінчуючи прикладним (відповідно до рівнів моделі OSI).

Діаграма класів для його опису представлена на рисунок 2.5.

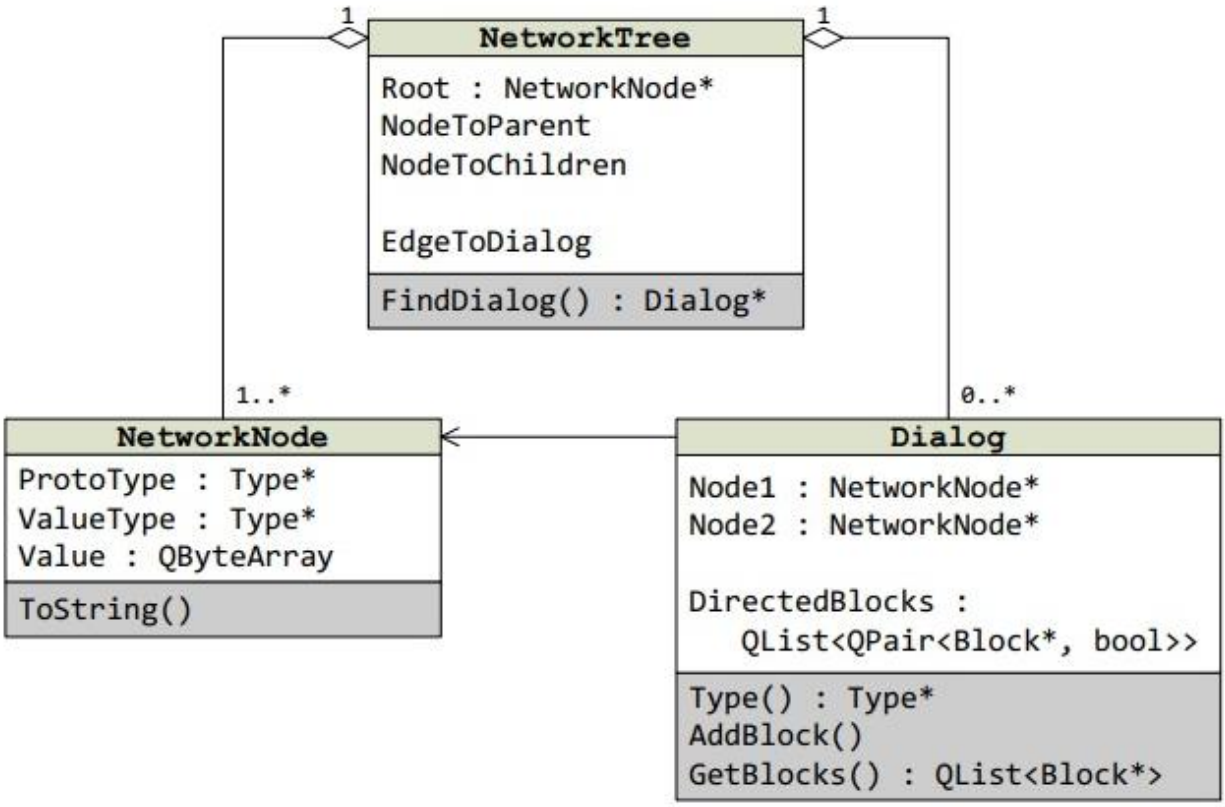

Рисунок 2.6 – Діаграма класів опису учасників мережевого обміну

Зв’язування рівнів довільного стека протоколів. Для забезпечення розбору пакетів довільного стека протоколів вводиться поняття распознавателя. Розпізнавач - це функція, яка за даними блоку і, можливо, деякої додаткової інформації визначає тип розбору цього блоку. Зазвичай функціональність по розпізнаванню потрібно для полів в заголовках протоколів, що допускають дані різних форматів. Наприклад, корисне навантаження в пакеті протоколу IPv4 може містити пакети протоколів TCP, UDP, GRE, ICMP і ін. Службове поле, що визначає протокол вищого рівня, міститься в IPv4-заголовку. Таким чином, для визначення типу розбору корисного навантаження використовуються дані блокародітеля. Відповідну категорію распознавателей будемо називати розпізнавачів полів. Для визначення типу розбору відновлених потоків використовується інша категорія распознавателей - розпізнавачі потоків з відомим типом збірки. Розпізнавач отримує на вхід вміст накопичувального буфера потоку і

інформація про тип розбору блоків, дані яких були додані в цей накопичувальний буфер. Розпізнавачі, які беруть на вхід тільки дані буфера, будемо називати універсальними. Такі розпізнавачі застосовуються до блоку-потоку, якщо його тип збірки невідомий, а також блоку-фрагменту, за умови, що за допомогою распознавателей полів не вдалося визначити тип розбору.

2.2 Метод відновлення потоків

При використанні такого алгоритму не виникає невизначеності, до якої одиниці передачі даних (PDU) слід віднести пакет, отриманий в неправильному порядку: рішення цього питання відкладається до тих пір, поки не буде оброблений безпосередньо передує пакет, або з’ясується, що цей пакет в трафіку відсутня. Запропонований метод вимагає більшої кількості фізичної пам’яті в порівнянні зі спробою визначити межі PDU відразу (в момент розбору пакету), однак результат аналізу стає більш точним.

2.3 Висновки до розділу 2

Подання розібраного мережевого пакету у вигляді дерева блоків, а також механізм распознавателей дозволяють аналізувати трафік вкладених тунелів будь-якої конфігурації. Завдяки заміщує блокам-фрагментам описуються модифікації (дешифрування, розпакування) даних. Введені поняття контексту і ключової групи дозволяють єдиним чином описувати розбір пакетів протоколів зі збереженням стану, а також виділяти безлічі пакетів для подальшого відновлення потоків. Розроблений алгоритм відновлення потоків даних стійкий до втрати і переупорядочивания окремих пакетів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АНАЛІЗА ТРАФІКУ

З урахуванням сформульованих в підрозділі 1.2 вимог була розроблена архітектура системи, що дозволяє обходитися єдиним комплектом вихідних кодів розбирачів протоколів, застосовуючи їх як в online-, так і в offline-режимі. Запропонована модель представлення даних реалізована у вигляді ядра системи, яке компілюється в динамічну бібліотеку. На базі API ядра побудована робота модулів розбору і розпізнавання даних. Було розроблено два інструменти - для проведення online- і offline-аналізу відповідно, - причому обидва використовують єдину інфраструктуру (рисунок 3.1). Залежно від встановлених параметрів проводиться збірка ядра і модулів розбору або для offline-аналізатора, або для online-аналізатора. Побудована за таким принципом система дозволяє в повній мірі використовувати переваги offline аналіза при розробці (налагодженні) модулів розбору і розпізнавання та подальшої експлуатації цих модулів (використовується той же вихідний код) при роботі online. Головна відмінність між offline- і online-версіями ядра полягає в механізмі управління пам’яттю: при роботі на нескінченному потоці даних необхідно гарантувати своєчасне звільнення пам’яті. Інструмент аналізу мережевих трас, в свою чергу, потребує розвиненому графічному інтерфейсі, можливості якого націлені на автоматизацію дій користувача при вирішенні конкретних класів задач, таких як: локалізація і деталізація мережевих з’єднань, зворотна інженерія протоколів (включаючи налагодження розбирачів), інтерактивне управління порядком розбору пакетів і ін. Система є модульною, як і більшість аналізаторів мережевого трафіку: для кожного протоколу створюється окремий модуль, в якому локалізована функціональність по роботі з цим протоколом. Кожен з інструментів використовує свій комплект модулів розбору. Важливо, що при компіляції розбирачів для різних підсистем (інструментів) використовується один і той же вихідний код. В рамках розробленої системи модулі розбору

також є незалежними: при додаванні нових розбирачів не потрібно вносити зміни в існуючі. Така незалежність досягається завдяки механізму зв’язування розбирачів, заснованому на застосуванні распознавателей.

База підтримуваних протоколів розширюється не тільки за допомогою ручної розробки коду розбирачів, але і шляхом автоматизованого перенесення (портування) розбирачів системи Wireshark. Для функцій API запропонованої системи і системи Wireshark задається відображення, вихідний код транслюється в проміжне представлення, на основі якого генерується перенесений код.

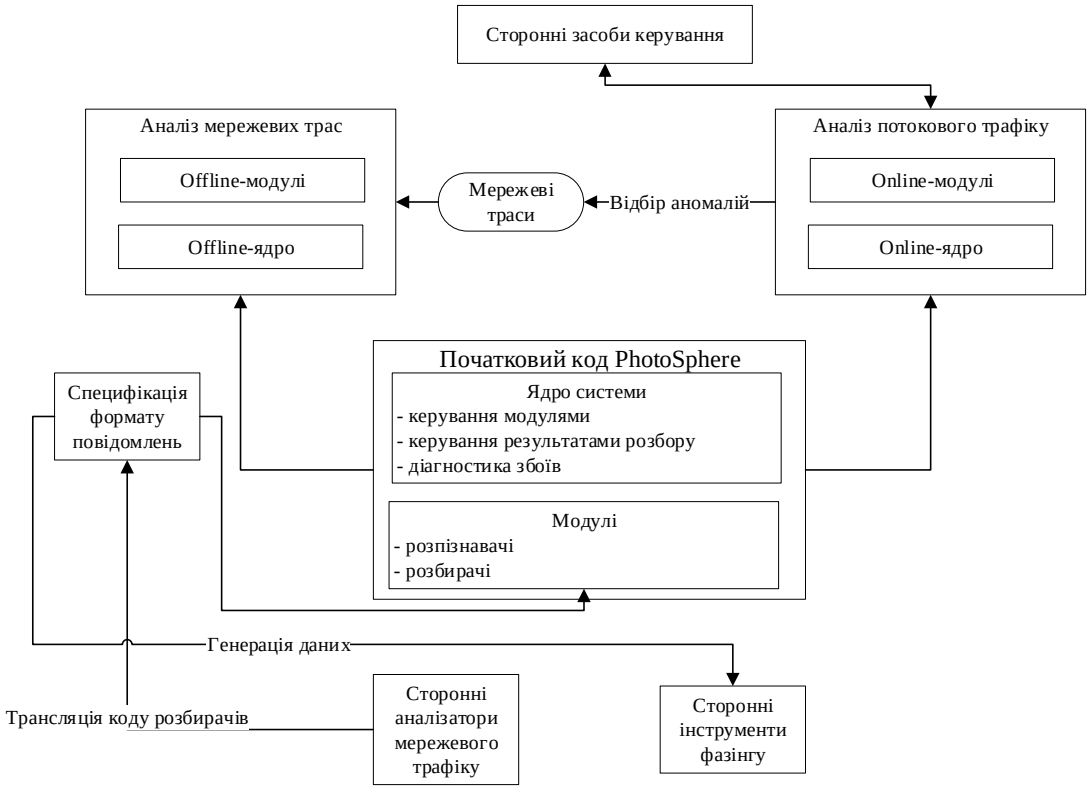

Рисунок 3.1 – Схема взаємодії програмних модулів

3.1 Проведення аналізу даних

Аналіз складається з послідовного виконання операцій розпізнавання і аналізу інформації. Для деяких протоколів відповідно до їх семантикою проводиться відновлення потоків, дані яких в подальшому також

розбираються. Таким чином, забезпечується послідовне підвищення рівня представлення даних (рисунок 3.2). В результаті користувач оперує над високорівневими даними, що потенційно підвищує якість проведеного ним аналізу.

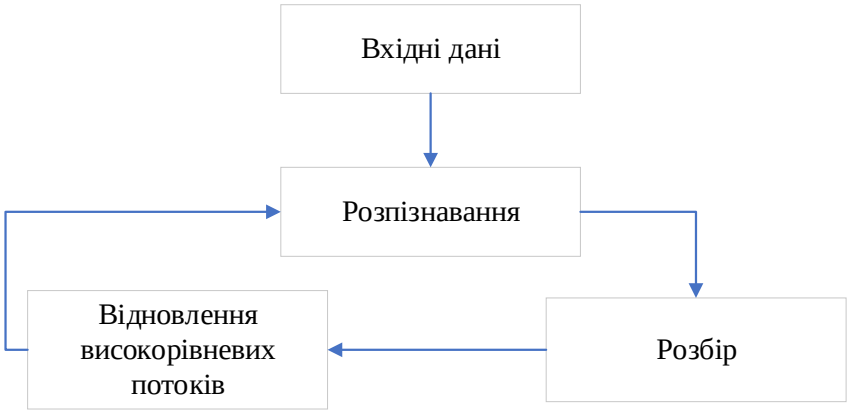

Рисунок 3.2 – Послідовне підвищення рівня представлення даних

На рівні ядра стан розбору описується за допомогою трьох стеків:

- стек блоків;
- стек контекстів (елемент - пара контекст і ключова група);
- стек мережевих вузлів (елемент - пара з відправника і одержувача).

Стек блоків застосовується для побудови дерева розбору: вершина стека стає батьком для блоків, створених розбирачем. Розпізнавачі застосовуються послідовно до першого успіху. Залежно від того, який блок аналізується, застосовуються різні групи розпізнавачів:

а) блок-потік:

- розпізнавачі потоків з відомим типом збірки;
- універсальні розпізнавачі;

б) складовий або одиночний блок:

- розпізнавачі полів;
- універсальні розпізнавачі.

За допомогою стека контекстів здійснюється угруповання блоків

(будується дерево контекстів): вершина стека характеризує протокол, заголовок якого розбирається в даний момент. Будемо називати контекст і ключову групу в вершині стека активними. Контекст поміщається в стек безпосередньо перед розбором блоку за умови, що тип розбору володіє контекстом (виставлений прапор ContextExistence), і виштовхується з стека після закінчення розбору. Перед початком аналізу створюються і поміщаються в стек фіктивні контекст з невизначеним типом і ключова група з порожнім ключем: завдяки цьому в кожен момент часу визначено активні контекст і ключова група. Фіктивна ключова група стає батьківській по відношенню до першого справжнього контексту. Перед додаванням контексту в стек перевіряється, чи існує в активній ключовий групі дочірній контекст відповідного типу. Якщо так, то новий контекст не створюється, а в стек поміщається вже існуючий. При створенні контексту проводиться ініціалізація розширення - спеціалізованих атрибутів, що описують стан з’єднання.

Врозширенні контексту протоколу SSL, зокрема, зберігається ключ шифрування, завантажений в систему користувачем з метою подальшого використання. Якщо контекст поміщається в стек перед розбором блоку, то ключова група визначається вже в процесі розбору. Виняток становить випадок з ключем нульового розміру GroupKeySize: в рамках контексту може бути сформована тільки одна група (зокрема, нульовий розмір ключа зафіксований для протоколу HTTP). Стек мережевих вузлів використовується для побудови NetworkTree. Якщо протокол передбачає наявність відправника і одержувача, то при розборі заголовка на стек поміщаються потрібні мережеві вузли. Журнал призначений для реєстрації помилок розбору. Кожен запис в ньому характеризується часовою позначкою, текстовим повідомленням, а також блоком, розбір якого привів до виникнення цього запису. Записи журналу також групуються за темами: прив’язка до теми здійснюється за допомогою типу розбору блоку. У журналі підтримується стек тем: перед початком аналізу в нього поміщається тема за

замовчуванням. Якщо тип розбору блоку передбачає наявність теми, то відповідна тема поміщається в стек безпосередньо перед його розглядом і виштовхується з стека після розбору. Якщо тема для типу не визначена, відповідний запис асоціюються з темою, що знаходиться на вершині стека. Як правило, одна тема задається для кожного модуля розбору. Аналіз файлу з трафіком завершується, коли всі блоки-потоки розібрані. У режимі online аналіз припиняється по команді користувача.

3.2 Ядро системи

API системи дозволяє аналітику реєструвати типи (описувати разборщікі) і розпізнавачі, створювати блоки-потоки, одиночні і складові фрагменти, управляти відновлення потоків. Зокрема, для коректного відновлення потоку з переупорядочение пакетів передбачена функція об’єднання дво блоків-потоків в один.

У таблиці 3.1 наведені основні функції, які використовуються при розробці розбирачів. Відзначимо, що модулі розбору повністю незалежні між собою. Ключ для формування групи будується при розборі заголовка: для активації групи застосовується функція activateKeyGroup. В процесі розбору використовуються і змінюються дані розширень активних контексту (функція contextExtension) і групи (функція keyGroupExtension). Передбачено додавання даних блоків-потоків, одиночних і складових фрагментів в потоки (функція streamAppend). При реєстрації распознавателя поля необхідно вказати ім’я відповідного поля, а також тип розбору блоку-батька. Таким чином, розпізнавачі, локалізовані в різних модулях, можуть належати до одного і того ж полю. Завдяки такому способу прив’язки з’являється можливість розширення функціональності без внесення змін до вже існуючого код модулів розбору. Подання в заданому форматі результатів аналізу трафіку.Інтерфейс побудови (таблиця 3.2) призначений для збереження декодованих даних в форматі, який визначається користувачем.

Формат задається шляхом створення модуля побудови.

Таблиця 3.1 – API для розробки модулів розбору

Ім’я функції	Опис
Операції над блоками	
processSingle	Створити та розібрати single-блок
processComposite	Створити та розібрати composite-блок
createStream	Створити блок-поток
completeStream	Виконати розбір блоку-потоку
streamAppend	Додати дані блока в блок-поток
Керування станом розбору	
contextExtension	Отримати дані розширення активного контексту
activateKeyGroup	Активувати в рамках поточного контекста групу
keyGroupExtension	Отримати дані розширення активної групи з ключем
setSrcDst	Активувати відправника та отримувача
Регистрація розбирачів і розпізнавачів	
regType	Зареєструвати тип
regRecognizer	Зареєструвати розпізнавач
getType	Отримати тип, який зареєстрований в іншому модулі
Журнал повідомлень	
log	Записати повідомлення

Зазвичай такі модулі застосовуються при проведенні online аналізу. Зокрема, можуть бути збережені відновлені PNG-, JPEG-, HTML-файли,

HTTP-потоки і т.д. Підтримується два способи прив’язки модулів побудови:

- по типу розбору блоку;

- по стеку мережевих вузлів.

При реєстрації модуль побудови повідомляє ядру системи тип блоків, які слід йому передавати. У разі прив’язки по стеку мережевих вузлів разом з типом блоку перевіряється вміст стека відправників і одержувачів: тільки при збігу блок буде відправлений у відповідний модуль.

Таблиця 3.2 – API для розробки модулів побудови

Ім’я функції	Опис
createBuffer	Створити буфер
completeBuffer	Зберегти буфер до файлу
bufferAppend	Додати дані блока в буфер у відповідності до заданого формату

Інструмент портування розбирачів з Wareshark.

Для збільшення бази підтримуваних протоколів системи був розроблений інструмент портирования розбирачів аналізатора Wireshark. Процес перенесення модуля розбору складається з трьох етапів:

- трансляція вихідного коду розбирача в проміжне представлення;
- модифікацію проміжного подання.

Генерація коду на основі модифікованого проміжного представлення. Код модулів розбору Wireshark написаний на мові CI. Розробка розбирачів, використовуваних описуваної системою, ведеться в рамках мови CI ++. Оскільки мови CI і CI ++ багато в чому схожі, як проміжного представлення, використовуваного при портуванні, було вибрано абстрактне синтаксичне дерево (АСД). АСД будується за допомогою інструменту Elsa, що є частиною системи статичного аналізу Oink. Вибір інструмента обумовлений простотою його застосування, а також наявністю у нього програмного інтерфейсу на мові CI ++. Перший етап полягає в побудові АСД по вихідного коду модуля

розбору Wireshark. На другому етапі проводиться модифікація побудованого АСД:

- заміна використовуваних типів і імен типів, якщо вони різні;
- виділення частини АСД, відповідної коду модуля (відділення частини АСД, відповідної включаються файлів);

Збір інформації, необхідної для проведення наступних етапів;

- заміна викликів функцій одного інтерфейсу на виклики функцій іншого інтерфейсу, заміна змінних і параметрів функцій;
- видалення непотрібних інструкцій, перевірка на нереалізовані перетворення.

Третій етап полягає в генерації цільового коду по АСД. При цьому, деякі ділянки коду (зокрема, специфічні для цільової системи функції) можуть бути згенеровані безпосередньо, без додавання в АСД. Перед проведенням трансляції код модуля піддається передобробці з метою збереження інформації про типи (розбору). Кордон між заголовними файлами бібліотеки Wireshark і кодом самого модуля позначається спеціальної pragma-директивою для того, щоб в подальшому відокремити код включення файл від коду модуля на рівні АСД. Потім код модуля обробляється стандартним системним препроцесором, після чого за допомогою інструменту Elsa проводиться трансляція препроцесірованного коду в АСД.

Для збору інформації про типи полів розбирача Wireshark використовується список ініціалізації масиву структур типу hf_register_info в АСД. Для отримання інформації про реєстрованих протоколах аналізуються функції proto_register_protoname і proto_reg_handoff_protoname, де protoname - назва протоколу. Локалізуються виклики наступних функцій:

- proto_register_protocol - реєстрація протоколу;
- create_dissector_handle - створення описателя розбирача;
- register_dissector - присвоєння імені розбирачів;
- find_dissector - повернення описателя по імені;

- dissector_add_uint - зв’язування розбирача в таблиці прив’язки.

Далі проводиться заміна викликів функцій ядра розбору Wireshark на виклики функцій ядра розбору розробленої системи у функціях, які виробляють розбір. Разборщікі в Wireshark приймають три або чотири параметри:

- покажчик на об’єкт буфера;

- покажчик на структуру packet_info, що містить значення набору службових полів, виділених в заголовках цього пакета раніше;

- покажчик на поддерево дерева розбору, в яке разборщик буде додавати дані;

- необов’язковий призначений для користувача параметр: його значення визначається угодою між що викликають і викликуваним розбирачів.

Функція-робирач розробленої системи приймає 2 параметра: покажчик на буфер пам’яті і розмір цього буфера.

Замінні функції можна розділити на групи:

- функції звернення до даних;

- функції додавання полів в дерево розбору;

- функції розбору даних, переданих за допомогою TCP-сегментів.

Функції звернення до даних перетворюються в пряме звернення до даних з використанням функцій для перетворення порядку байтів. Функції додавання полів в дерево розбору замінюються функціями обробки простого або складеного блоку-фрагмента з відповідним типом розбору. Функція tcp_dissect_pdus призначена для поділу TCP- потоку на повідомлення, межі яких можуть не збігатися з межами сегментів. Так як в розробленій системі збірка TCP-потоків проводиться модулем TCP, то в розбирачів пакетів верхніх протоколів необхідно реалізувати лише розбиття даних потоку на повідомлення. Тому виклик функції tcp_dissect_pdus замінюється послідовністю інструкцій, які виробляють це розбиття. Новий вихідний код генерується за АСД за допомогою інструменту Elsa.

3.3 Розроблені аналізатори

Інструмент online-аналізу призначений для отримання даних з трафіку в режимі реального часу: він повинен працювати безперервно з продуктивністю, достатньою для розбору пакетів, що надходять на мережевий інтерфейс (потенційно нескінченний вхідний потік даних). Інструмент складається з трьох компонентів:

- модуль M1 здійснює взаємодію з мережевим інтерфейсом і зберігає вступники пакети в кільцевої буфер;

- модуль M2 виконує розбір пакетів кільцевого буфера і витягує необхідні дані;

- модуль M3 зберігає витягнуті дані в файли на жорсткому диску (формат файлу визначається модулем побудови).

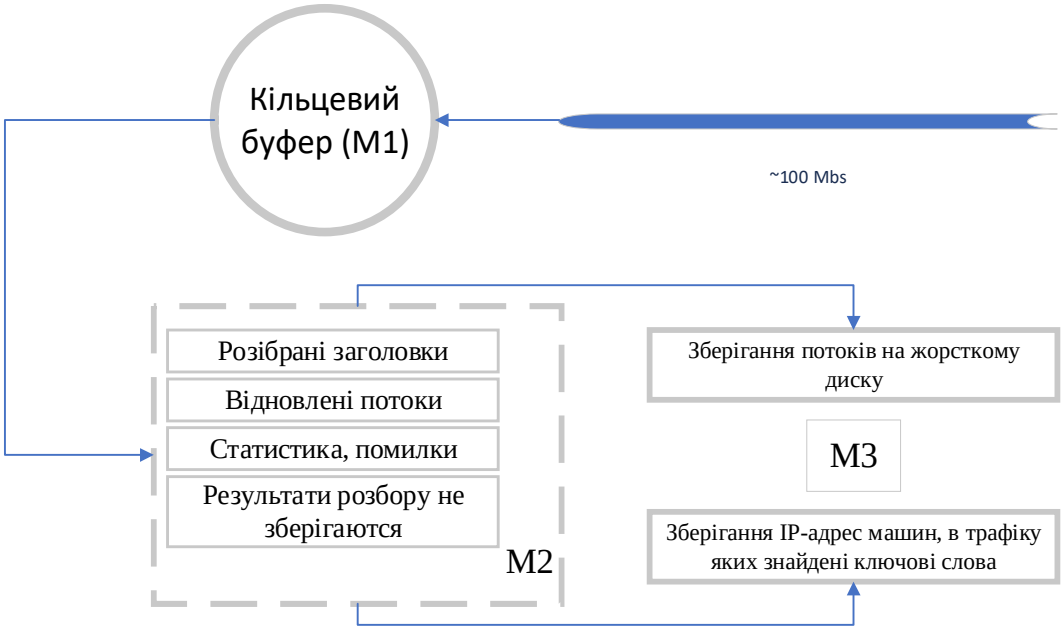

Рисунок 3.3 – Компоненти online-аналізатору

На даний момент компонент M1 може отримувати пакети або за допомогою API бібліотеки PCAP, або за допомогою ZeroMQ-сокета. Пакети послідовно записуються в кільцевий буфер RingBuffer, після чого

аналізуються компонентом M2. При розборі потенційно нескінченного потоку даних необхідно явно обмежувати розмір доступної аналізатору фізичної пам’яті: в іншому випадку вільна пам’ять закінчиться, і аналізатор в кращому випадку завершить роботу аварійно. RingBuffer займає фіксований обсяг у фізичній пам’яті, який вираховується перед початком проведення аналізу. У процесі аналізу пам’ять використовується для зберігання потоків, відновлення яких ще на завершилося, і, як наслідок, дерева контекстів. Кількість таких потоків і вершин в дереві контекстів може необмежено зростати: необхідне введення граничних значень. Запропоновано наступна стратегія звільнення пам’яті: при досягненні порогового значення видаляти першими об’єкти (потоки, ключові групи, контексти), які довше за інших не оновлюється. Така стратегія реалізована за допомогою розробленого generic-контейнера PriorityStorage. Online-аналізатор окремо зберігає в файл пакети, розбір яких відбувається з помилками. Під помилкою в даному випадку розуміється невідповідність фактичних даних опису протоколу в модулі розбору. Якщо кількість таких помилок перевищує задане граничне значення, велика ймовірність того, що відповідний модуль розбору містить неточності. В такому випадку необхідно проаналізувати збережені пакети за допомогою offline-аналізатора, після чого внести в код модуля розбору (загальний для двох інструментів) необхідні правки. Інструмент online-аналізу може використовуватися сторонніми інструментами для отримання даних в заданому форматі (див. Інтерфейс побудови). Передбачений інтерфейс отримання результатів розбору через ZMQ-сокет. Результати розбору мережевих пакетів можуть використовуватися в різних цілях, зокрема, для перевірки здійсненності правил, що описують політику безпеки.

3.3.1 Розподіл ресурсів

При проведенні аналізу в режимі online відбувається поступове збільшення числа контекстів, ключових груп, потоків. Щоб не виходити за

рамки фізичної пам’яті, відведеної аналізатору, необхідно перерозподіляти її між об’єктами, які з великою ймовірністю більше не будуть потрібні (можуть бути видалені), і об’єктами, з якими ще належить працювати (видалення небажано). Для цього використовується розширена чергу з пріоритетом та фіксованого місткістю PriorityStorage.

Розширена чергу з пріоритетом.

PriorityStorage PriorityStorage фактично являє собою кеш з можливістю виконання деякої функції над елементом безпосередньо перед його видаленням: виконати розбір потоку, видалити дочірні ключові групи контексту. Елементи черзі впорядковані за часом поновлення: оновлення елемента має на увазі максимізацію його пріоритету. Для контексту і ключової групи оновлення відбувається при активації (тобто всякий раз, коли вони опиняються на вершині стека контекстів). Оновлення потоку має на увазі або його потрапляння на вершину стека блоків, або додавання в його буфер нової порції даних. Місткість (capacity) черги задається при створенні, однак може бути змінена, коли контейнер порожній. Слід відрізняти від місткості розмір черги, що дорівнює кількості елементів в ній. При цьому розмір черги не перевищує її місткості. Всі елементи в контейнері різні, дублікати не допускаються. Найбільшим пріоритетом має останній доданий в контейнер елемент або елемент, для якого виконана операція максимізації пріоритету. Підтримуються наступні операції:

- додавання елемента (за значенням);
- максимізація пріоритету елемента (за значенням);
- отримання / витяг елемента з мінімальним пріоритетом;
- витяг довільного елемента (за значенням).

Контейнер реалізований з використанням словника M і двухзв’язного списку L: M використовується для швидкого доступу до елементів, L - для підтримки впорядкованості відповідно до оновленнями. M задає відображення безлічі елементів контейнера на безліч елементів L. Кожен елемент L зберігає інформацію про попередній і наступному (за пріоритетом)

елементах L, а також копію елемента контейнера (елементи контейнера повинні допускати копіювання). У PriorityStorage також міститься інформація про першому і останньому елементах L.

Кільцевій буфер RingBuffer організований у вигляді списку блоків пам’яті фіксованого розміру. У кожен момент часу два блоки в буфері помічені: writable - блок для запису і readable - блок для читання. Дані (мережеві пакети) записуються в writable-блок послідовно доти, поки в ньому досить вільного місця. Коли місце закінчується, відповідна позначка переноситься на наступний за списком блоку. З плином часу все блоки будуть вичерпані і почнеться перезапис. Будемо називати поколінням блоку пам’яті кількість перезаписів його нульового байта: з кожним перезаписом покоління иінкрементується. При запису даних в буфер запам’ятовується покоління writable-блоку: якщо при подальшому читанні з’ясується, що його покоління змінилося, дані будуть проігноровані. Так здійснюється контроль за цілісністю оброблюваних даних. Операції читання і запису (точніше, перенесення позначок) є потокобезпечні. Якщо потрібно прочитати дані writable блоку, то черговий запис даних незалежно від наявності вільного місця буде проведений в наступному за списком блоку пам’яті. У режимі реального часу важлива висока швидкість обробки: якщо мережеві пакети надходять швидше, ніж обробляються, то проводити аналіз безглуздо. Операція копіювання даних займає чимало часу, тому при читанні з буфера не виконується - замість цього блок пам’яті, що містить потрібний пакет, блокується на читання. Для чергового запису буде обраний наступний за списком блоку.

3.3.2 Offline-аналіз мережевих трас

Інструмент offline-аналізу головним чином призначений для розробки і налагодження модулів розбору: з його допомогою аналітик отримує максимально деталізовану "картину того, що відбувається" в мережевій трасі

із зазначенням виниклого в процесі роботи над помилками. Інструмент має графічний інтерфейс, що дозволяє простежити за тим, як відбувається відновлення потоків високорівневих даних (наприклад, файлів формату PNG або PDF) з мережевих пакетів. Також передбачені механізми пошуку по трафіку і навігації між різними уявленнями результатів розбору.

Центральне місце в інструменті offline-аналізу займають компоненти відображення результатів розбору:

- дерево блоків; журнал помилок розбору;
- граф мережних взаємодій;
- тимчасова діаграма.

3.3.3 Отримання результатів розбору

Дерево блоків являє собою результат розбору одного блоку-потоку: корінь описує блок-потік, інші вершини - виділені блоки-фрагменти (мережеві пакети і поля в них). Дерево блоків синхронізовано з компонентом відображення буфера даних: при виділенні блоку в дереві відповідні йому байти підсвічуються. Для блоку-потоку можна переглянути список блоків, дані яких сформували його накопичувальний буфер. При цьому блоки, порядок яких був змінений при передачі, будуть позначені (рисунок 3.4).

Основним засобом представлення мережевої траси в Wireshark є розібрані пакети у вигляді списку: при цьому тільки для одного виділеного пакету відображається повний стек протоколів і значення полів в заголовках цих протоколів. Пакет, як елемент списку, представляється за допомогою рядка, що складається з значень фіксованого набору полів, виділених в заголовку протоколу мережевого рівня (IP-адреси), а також полів заголовка протоколу найвищого рівня, який вдалося розібрати. Фиксованість уявлення пакетів може викликати труднощі у багатьох випадках, наприклад, при аналізі тунельних протоколів.

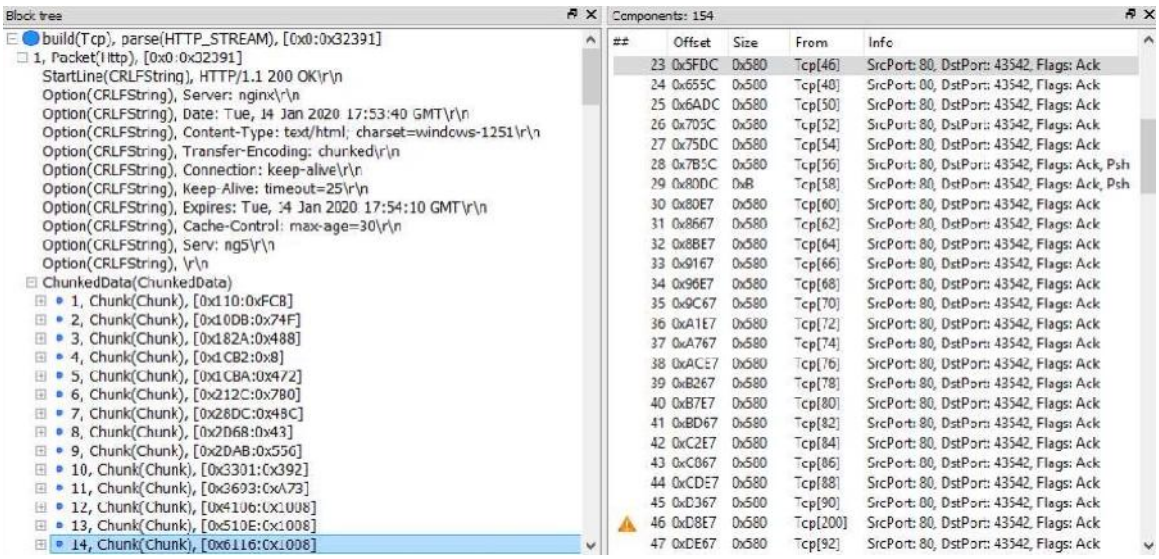

Рисунок 3.4 – Відображення дерева блоків та списку компонентів

При інкапсуляції виду IP-in-IP кожен мережевий пакет буде містити по два заголовка протоколу IP. У списку пакетів інструмент Wireshark відобразить поля останнього (верхнього) IP заголовка, який з точки зору тунельного з'єднання не є репрезентативним (рисунок 3.5): щоб зрозуміти, який хост з внутрішньої мережі ініціював взаємодію, аналітику доведеться виділяти кожен пакет і перевіряти значення поля мережевого адреси нижчого IP-заголовка. При використанні дерева блоків таких труднощів, очевидь, не виникає.

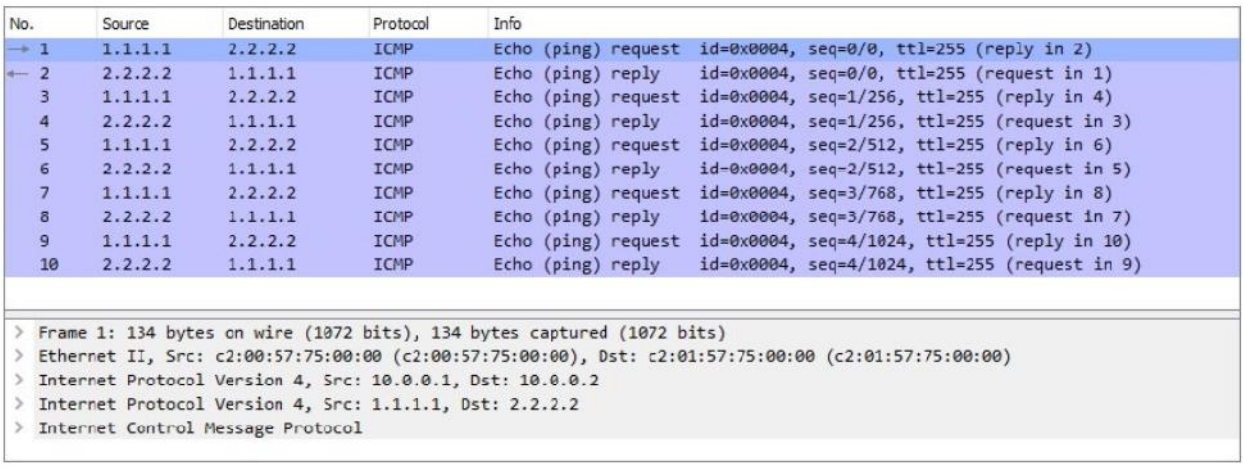

Рисунок 3.5 – Відображення інкапсуляції IP-in-IP

Компонент відображення журналу синхронізований з деревом блоків: це дозволяє швидко локалізувати блок, розбір даних якого привів до виникнення повідомлення (рисунок 3.6). Підтримується фільтрація записів по потокам і темам повідомлень.

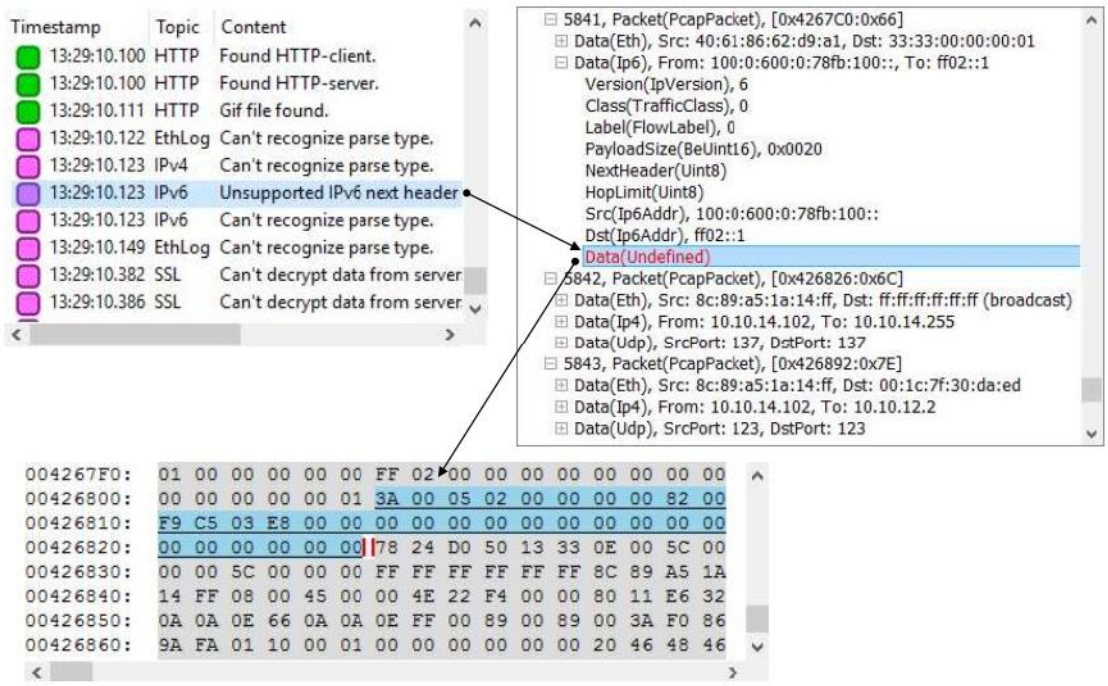

Рисунок 3.6 – Налагодження розбирача з використанням журналу

При розслідуванні інциденту порушення інформаційної безпеки необхідно локалізувати мережеві з’єднання, за допомогою яких цей інцидент виник і розвивався в часі: аналітик повинен володіти деяким критерієм (або безліччю таких критеріїв) на вміст мережевих пакетів. Одним з підходів до вирішення завдання локалізації є уявлення мережевих взаємодій за допомогою графа, в якому вершин відповідають боку мережевого взаємодії, а ребра відображають факт взаємодії і, можливо, деякі його характеристики (наприклад, такі як інтенсивність. При цьому одна і та ж сторона може брати участь відразу в декількох взаємодіях. Далі потрібно провести детальний аналіз виділених з’єднань: простежити за порядком відправлення / отримання пакетів; переглянути значення цікавлять полів. Розглянуті в розділі 1

інструменти не надають графічних компонентів для роботи з таким сценарієм в інструменті offline-аналізу реалізовано два способи візуалізації мережевих взаємодій:

- граф кінцевих вузлів (Endpoints);
- граф, деталізує мережеві взаємодії обраного кінцевого вузла (Nodes).

Граф Endpoints відображає мережеві з’єднання між вузлами, що характеризуються однаковими ланцюжками протоколів: при цьому можна вибрати протокол, взаємо дії в рамках якого будуть показані. Вершини графа відповідають мережевим вузлам. Є ребра двох видів (рисунок 3.7): одні для подання стека нижчих заголовків (пунктирні), інші для відображення мережевих взаємодій між вузлами (суцільні).

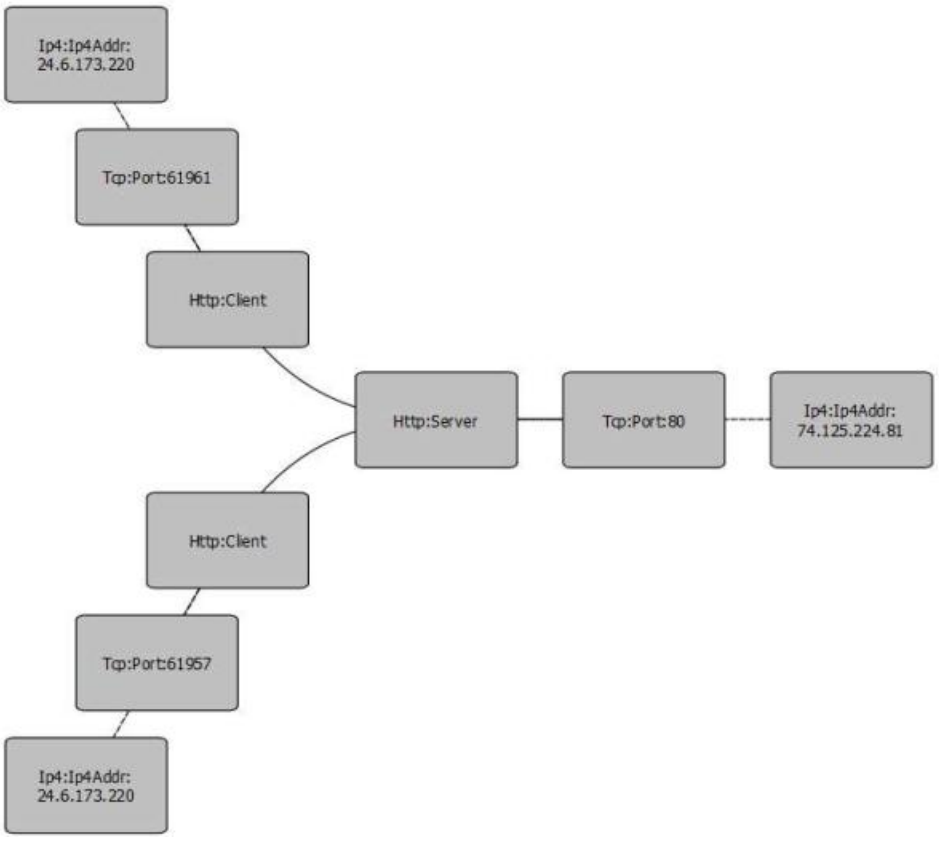

Рисунок 3.7 – Приклад графу Endpoints

Граф Nodes деталізує взаємодії обраного на графі Endpoints кінцевого вузла: для кожного взаємодії відображається весь стек мережевих протоколів

(рисунок 3.8). Передбачена можливість фільтрації графів у вмісті мережевих пакетів, в тому числі за допомогою регулярних виразів.

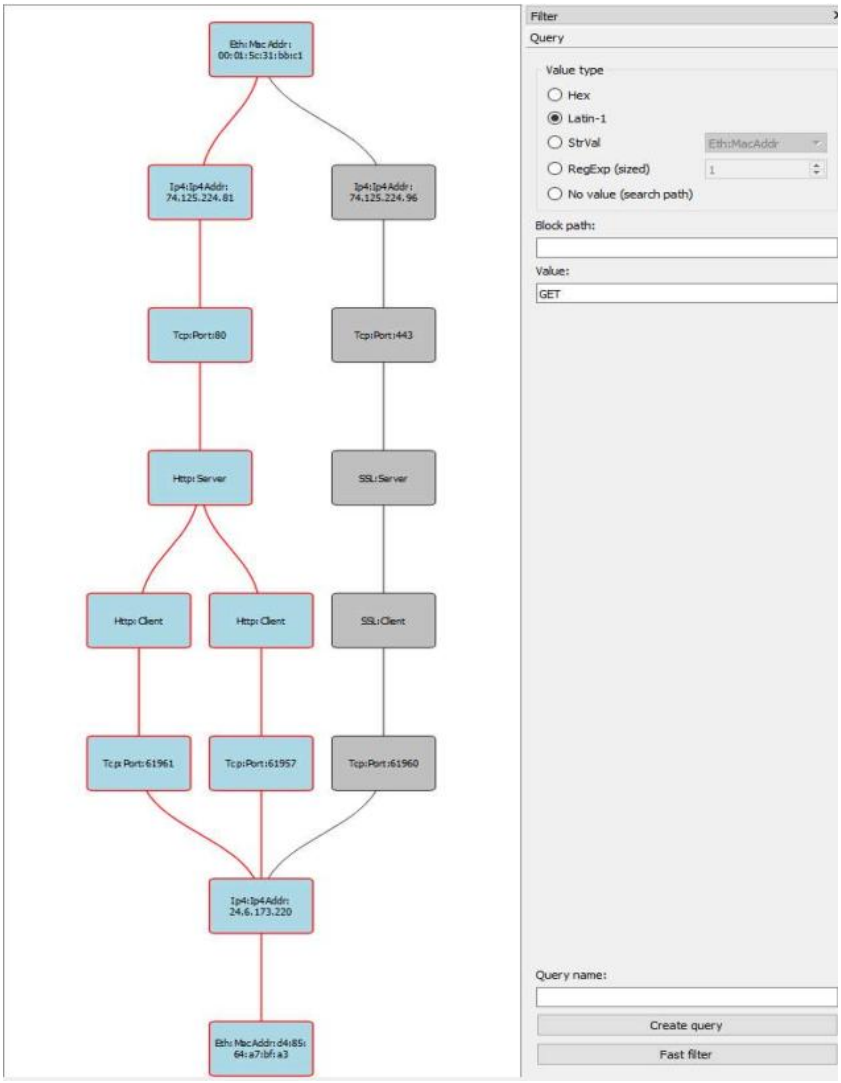

Рисунок 3.8 – Приклад використання фільтру Nodes

Для проведення детального аналізу окремого мережевого взаємодії пропонується тимчасова діаграма, де кожен пакет відображається у вигляді стрілки із зазначенням відправника і одержувача (рисунок 3.9). При цьому можна вказати, які заголовки протоколів і які поля в них повинні відображатися над стрілками: аналітик адаптує графічний компонент під свої потреби. Слід зазначити, що в аналізаторі Wireshark є аналогічний компонент, однак в ньому відсутня можливість настройки полів, значення

яких будуть відображатися для кожного пакету: це може викликати труднощі, якщо потрібне поле відсутнє, або приводити до перевантаженості відображення, якщо полів занадто багато.

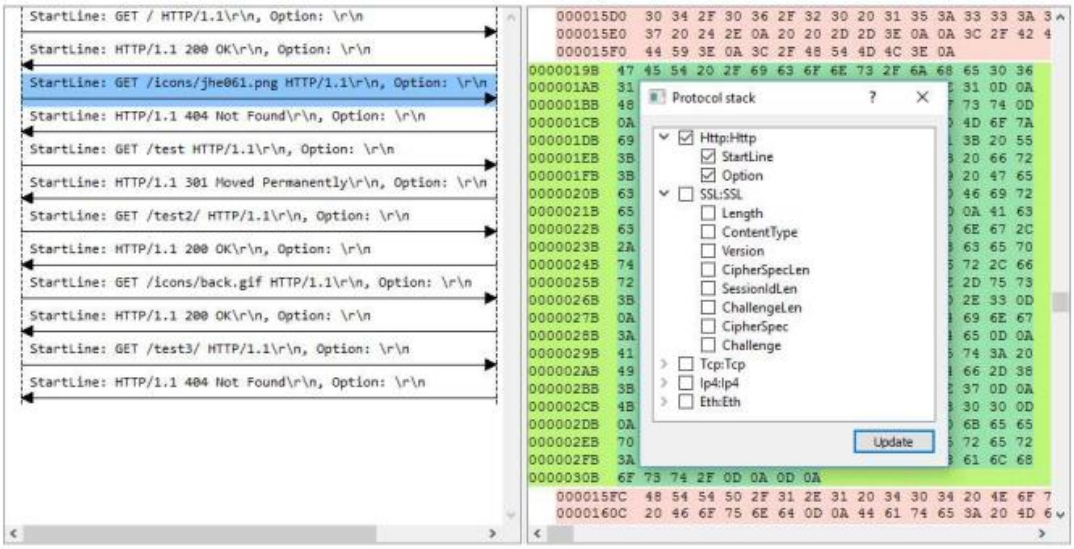

Рисунок 3.9 – Вибір полів для відображення на часовій діаграмі

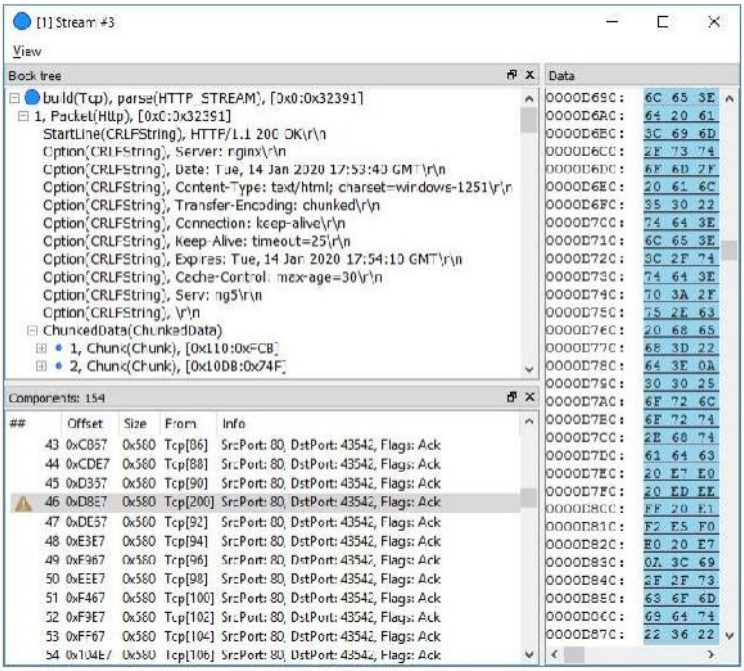

Рисунок 3.10 – Аналіз відновлення потоку даних з переупорядкуванням TCP-пакетів

Для траси Trace14.pcap демонструється застосування розробленого методу відновлення потоків. Аналіз показав, що до відновленого TCP-потоку (#3) відноситься 154 TCP-пакета (рисунок 3.10). У першому стовпчику вікна «Components» вказані порядкові номери, під якими в відновлений потік додані TCP-сегменти з пакетів з номерами, зазначеними в четвертому стовпці (це номери пакетів в мережевий трасі): наприклад, корисне навантаження 86-го TCP-пакета була додана під номером 43. Відзначимо, що TCP-пакети з меншими номерами в четвертому стовпці розбираються раніше за інших. Виходячи із значення поля Sequence Number TCP-заголовка корисне навантаження 92-го TCP-пакета не може бути додане в потік, частково відновлений на момент закінчення обробки пакета з номером 90. У результаті, створюється TCPпоток (# 4), в який додаються дані TCP сегмента 92-го пакета. TCPпотокі # 3 і # 4 будуть об’єднані в подальшому, після проведення розбору 200-го пакета. Подальший розбір відновленого TCP-потоку дозволив виділити HTTP пакети в ньому (вікно «Block tree») і витягти HTML сторінку.

ВИСНОВКИ

Проведено дослідження методів та засобів аналізу мережного трафіку. Проаналізовано методи відновлення потоків даних для протоколів довільного рівня, в тому числі прикладного, які стійкі до втрат окремих мережевих пакетів, а також їх переупорядкування. Запропонована архітектура системи поглибленого аналізу мережевого трафіку, що дозволяє розробляти і налагоджувати модулі підтримки протоколів на попередньо збереженому трафіку (offline) і згодом використовувати ці модулі в режимі online. Розроблені та реалізовано програмні інструменти для проведення поглибленого аналізу мережевого трафіку в online і offline режимах.

Представлена програмні засоби можуть успішно застосовуватися при аналізі трафіку як в online-, так і в offline-режимі. Запропонованийметод опису даних охоплює існуючі особливості передачі мережевого трафіку.

Для формального опису розбирачів застосовується мова C ++: аналітик безпосередньо реалізує функції розбору і розпізнавання даних. У той же час функції розбору можуть бути згенеровані автоматично на основі деякого уявлення структури повідомлення протоколу. Розробка і впровадження високорівневої декларативної мови опису формату дозволить прискорити процес розробки розбирачів.

Формальний опис структури також може використовуватися для генерації трафіку з заданими характеристиками. Подібна функціональність необхідна при вирішенні задачі забезпечення інформаційної безпеки за допомогою фаззінга

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дяченко В.О., Міхаль О.П., Сидоров Д.С. Методи аналізу мережного трафіку комп’ютерній мережі // Сучасні напрями розвиткуінформаційно-комунікаційних технологій та засобів управління. Матеріали десятої міжнародної НТК. –Баку: ВА ЗС АР; Харків: НТУ «ХПІ»; Харків: ДП«ПДПРОНДІАВІАПРОМ»; Жиліна: УмЖ, 2020. – 9-10 квітня 2020. – С. 86.

2. Mike Cloppert. An Overview Of Protocol Reverse-Engineering. [Електроний ресурс] https://digitalforensics.sans.org/blog/2012/07/03/an-overview-of-protocol-reverse-engineering, дата обращения 02.02.2017.

3. IETF RFC 791. J. Postel. Internet Protocol, September 1981.

4. Antonios Atlasis. Fragmentation (Overlapping) Attacks One Year Later, Troopers 13 – IPv6 Security Summit, 2013.

5. IETF RFC 793. J. Postel. Transmission Control Protocol, September 1981.

6. Judy Novak, Steve Sturges. Target-Based TCP Stream Reassembly, 2007.

7. Jon C. R. Bennett, Craig Partridge, Nicholas Shectman. Packet reordering is not pathological network behavior // IEEE/ACM Transactions on Networking (TON) archive, Volume 7 Issue 6, Dec. 1999, Pages 789-798.

8. IETF RFC 2616. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, June 1999.

9. А. Ализар. Доля зашифрованного трафика в интернете выросла в несколько раз. [Електроний ресурс] https://xakep.ru/2014/05/15/62500/, дата обращения 10.02.2017.

10. VPN Tunneling Protocols. [Електроний ресурс] https://technet.microsoft.com/en-us/library/6e3cd69cdc8c-483e-98bc-8d2e7e76e048, дата обращения 01.02.2017.

11. Мэйволд Э. «Межсетевые экраны» // Безопасность сетей: Информация, ИНТУИТ.

12. Hussain Ahmad Madni Uppal, Memoona Javed and M.J. Arshad. An Overview of Intrusion Detection System (IDS) along with its Commonly Used Techniques and Classifications // International Journal of Computer Science and Telecommunications, volume 5, Issue 2, 2014.

13. Christos Douligeris, Aikaterini Mitrokotsa. DDoS attacks and defense mechanisms: classification and state-of-the-art // Computer Networks, 2003.

14. Laura Chappell. Troubleshooting Tips and Tricks for TCP/IP Networks // SHARKFEST 11, Stanford University, June 13–16, 2011 15. P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012.

16. Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Автоматизация тестирования соответствия для телекоммуникационных протоколов // Труды Института системного программирования РАН, том 26, выпуск 1, 2014, стр. 109-148.

17. Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800- 94, 127 pages, February 2007.

18. Maurizio Dusi, Francesco Gringoli, Luca Salgarelli. IP Traffic Classification for QoS Guarantees: the Independence of Packets // In: Proceedings of The 1st IEEE International Workshop on IP Multimedia Communication (IPMC 2008), Augush 2008.

19. ГОСТ Р ИСО/МЭК 7498-1-99. – «ВОС. Базовая эталонная модель. Часть 1. Базовая модель». – ОКС: 35.100.70. – Действует с 01.01.2000. – 62с.

20. Christopher Parsons. Deep Packet Inspection in Perspective: Tracing its lineage and surveillance potentials // Working Paper, January 2009.

21. IETF RFC 768. J. Postel. User Datagram Protocol, August 1980.

22. IEEE Standard for Ethernet, 802.3-2012 – section one. 2012-12-28. p. 53. Retrieved 2014-07-06.

23. Xiaoming Zhou and Piet Van Mieghem. Reordering of IP Packets in

Internet // International Workshop on Passive and Active Network Measurement, PAM 2004, pp 237-246.

24. Arjuna Sathiaseelan and Tomasz Radzik. Improving the performance of TCP in the case of packet reordering // IEEE International Conference on High Speed Networks and Multimedia Communications, HSNMC 2004, pp 63-73

25. IETF RFC 5246. T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2, August 2008.