МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри комп'ютеризованих систем захисту інформації
_____ Михайло СТЕПАНОВ
«_____»_____ 2023 р.

На правах рукопису
УДК 004. 056:510.22(043.3)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Засіб захисту віддалених підключень до корпоративної мережі

Виконавець:	Максим ЛЕВАНДОВСЬКИЙ
Науковий керівник: к.т.н., доцент	Іван ПАРХОМЕНКО
Консультант розділу «Охорона навколишнього середовища»: к.т.н., доцент	Тетяна ДМИТРУХА
Нормоконтролер: к.т.н., доцент	Іван ПАРХОМЕНКО

Київ 2023

іНАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Магістр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«__» _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

здобувача вищої освіти Левандовського Максима Дмитровича

Тема: *Засіб захисту віддалених підключень до корпоративної мережі*

затверджена наказом ректора від «15» вересня 2023 р. № 1814/ст.

1. Термін виконання: з 16.10.2023 р. по 31.12.2023 р.
2. Вихідні дані: дослідити побудову архітектури корпоративної мережі та існуючі методи захисту віддалених підключень; на основі проведеного аналізу розробити програмний засіб захисту віддалених підключень.
3. Зміст пояснювальної записки: архітектура корпоративної мережі та існуючі методи захисту віддалених підключень; реалізація програмного засобу захисту віддалених підключень.

5.КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ з/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	16.10.2023	*Виконано*
2.	Аналіз літературних джерел	17.10.2023	*Виконано*
3.	Обґрунтування вибору рішення	20.10.2023	*Виконано*
4.	Збір інформації	25.10.2023	*Виконано*
5.	Аналіз існуючих архітектур корпоративних мережевих систем	30.10.2023	*Виконано*
6.	Дослідження та аналіз сучасних протоколів та алгоритмів захисту віддалених підключень	10.11.2023	*Виконано*
7.	Вибір технологій розробки	24.11.2023	*Виконано*
8.	Реалізація програмного засобу моніторингу та захисту мережевого трафіку у корпоративній мережі	7.12.2023	*Виконано*
9.	Перевірка на антиплагіат	13.12.2023	*Виконано*
10.	Оформлення і друк пояснювальної записки	20.12.2023	*Виконано*
11.	Оформлення презентації	8.12.2023	*Виконано*
12.	Отримання рецензій від рецензента	20.12.2023	*Виконано*

6. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона навколишнього середовища	Дмитруха Т.І.		

7. Дата видачі завдання: «16» жовтня 2023 р.

Здобувач вищої освіти Максим Левандовський
(підпис, дата)

Керівник кваліфікаційної роботи Іван Пархоменко
(підпис, дата)

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків і має 84 сторінки основного тексту, має 30 рисунків, 5 таблиць, 15 сторінок додатків. Список використаних джерел містить 30 найменувань і займає 3 сторінки. Загальний обсяг роботи складає 113 сторінок.

Метою роботи є розроблення засобу захисту віддалених підключень до корпоративної мережі.

У даній дипломній роботі проведено комплексне дослідження та розробку засобу захисту віддалених підключень до корпоративної мережі. Актуальність цього завдання обумовлена зростанням використання віддалених робочих місць та необхідністю забезпечення високого рівня безпеки під час доступу до корпоративних ресурсів.

У роботі проведено аналіз існуючих підходів до захисту віддалених підключень, визначено основні загрози та вразливості, що стосуються корпоративних мереж. На основі цього аналізу розроблено та впроваджено програмний модуль, який забезпечує ефективний захист віддалених підключень до корпоративної мережі.

Досягнуті результати реалізовано та протестовано на практиці, зокрема, шляхом встановлення та використання віддалених підключень до експериментальної корпоративної мережі. Засіб захисту продемонстрував високу ефективність та надійність у реальних умовах експлуатації.

Отримані результати дозволяють вирішити актуальну проблему захисту віддалених підключень до корпоративної мережі та внести позитивний вклад у розвиток сфери інформаційної безпеки корпоративних структур.

Ключові слова: корпоративна мережа, мережева аномалія, tun/tap інтерфейси, засіб захисту віддалених підлкючень та моніторингу корпоративної мережі, протоколи.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

АП	–	аналізатор пакетів;
КМ	–	корпоративна мережа;
МБ	–	модель безпеки;
ІТ	–	інформаційні технології;
ШНМ	–	штучна нейронна мережа;

ВСТУП

Актуальність. У сучасному світі, де все більше працівників працюють віддалено, захист корпоративної мережі від хакерів і інших загроз є критично важливим. Віддалені підключення можуть створювати ряд потенційних загроз для корпоративної мережі. Наприклад, хакер може використовувати віддалене підключення, щоб отримати доступ до конфіденційної інформації, наприклад, фінансових даних або патентів. Хакер також може використовувати віддалене підключення, щоб завантажити шкідливе програмне забезпечення на комп'ютери в корпоративній мережі. Існує ряд засобів захисту, які можуть допомогти захистити корпоративну мережу від загроз, пов'язаних з віддаленими підключеннями. Одним із таких засобів є використання засобів моніторингу трафіку мережі. В умовах постійної еволюції кіберзагроз та швидкого розвитку технологій, забезпечення безпеки інформаційних систем стає пріоритетним завданням для будь-якої організації. Розпізнавання та реагування на потенційні загрози стає ключовим елементом стратегії інформаційної безпеки.

Отже засіб захисту віддалених підключень до корпоративної мережі є необхідним й актуальним. Через це є необхідним розробка програмного модулю моніторингу мережевого трафіку який, в свою чергу, буде аналізувати трафік працівники і визначати його тенденцію, аномалії та формувати звіт на основі цієї інформації. Це дозволить своєчасно зреагувати та знешкодити потенційну загрозу, залишивши корпоративну мережу у безпеці.

Метою кваліфікаційної роботи є розробка засобу захисту віддалених підключень до корпоративної мережі. Щоб досягнути цієї мети необхідним є розв'язання таких **задач**:

1. Дослідження сучасної архітектури корпоративної мережі.
2. Проаналізувати існуючі методи та алгоритми захисту віддалених підключень до корпоративної мережі.

3. Розробити програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі.
4. Протестувати розроблений засіб моніторингу та захисту мережевого трафіку у корпоративній мережі.

Галузь застосування. Розроблений програмний модуль є універсальним і може бути успішно впроваджений у різноманітних галузях, де існує потреба у захисті та моніторингу критичної корпоративної інфраструктури.

Об'єкт дослідження: процес передачі трафіку, який проходить через корпоративну мережу під час віддаленого підключення.

Предмет дослідження: моделі та системи моніторингу та ідентифікації трафіку працівників під час віддалених підключень до корпоративної мережі.

Методи дослідження базуються на тенденції, кількості аномалій та загроз в трафіку корпоративного користувача, які визначаються за рахунок аналізів пакетів даних, виявлення аномалій, моделювання поведінки користувачів.

Основними науковими результатами роботи є: впровадження механізму визначення місця походження та ступінню небезпеки трафіку, що забезпечує додатковий шар аналізу та захисту. Також проведена модифікація алгоритмів, які динамічно адаптуються до змін у структурі та обсязі корпоративного трафіку, що робить засіб більш гнучким та ефективним у змінних умовах роботи мережі.

Практична цінність роботи полягає у тому, що застосунок надає ефективний механізм для виявлення потенційно шкідливих або несанкціонованих дій під час віддалених підключень, забезпечуючи високий рівень безпеки інформаційної інфраструктури. Забезпечення ефективного моніторингу та відслідковування дій працівників під час віддалених підключень, що сприяє виявленню ненормальних та непередбачуваних ситуацій.

Апробація. Основні положення роботи доповідалися та обговорювалися на конференції X International Conference Information Technology and Implementation ((Satellite), 21 листопада 2023 р.) та опубліковано статтю та тези доповіді:

– M. Levandovskiy, L. Myrutenko, “Modern methods and tools for reverse engineering to detect and circumvent cyber threats.", X International Conference Information Technology and Implementation (Satellite), Nov. 2023

РОЗДІЛ 1. АНАЛІЗ МЕТОДІВ ЗАХИСТУ ВІДДАЛЕНИХ ПІДКЛЮЧЕНЬ.

1.1. Архітектура та Засоби Захисту в Корпоративних Мережах

Корпоративна мережа - це технічна інфраструктура, яка об'єднує комп'ютери, сервери та додаткові електронні інструменти корпорації. Вона дозволяє працівникам працювати зі спільними ресурсами та забезпечує доступ до внутрішніх і зовнішніх додатків та ресурсів.

Велика компанія може мати розгалужену корпоративну мережу, яка з'єднує будівлі навколо головного офісу з високошвидкісним Інтернетом та іншими важливими послугами. Невеликі підприємства, які функціонують у локальних мережах (LAN), також можуть володіти корпоративними мережами.

Корпоративна мережа включає різні апаратні компоненти, зокрема маршрутизатори, комутатори, брандмауери та балансувальники навантаження. Ці апаратні компоненти підтримують програмне забезпечення, таке як операційні системи, сервери бази даних, сервери пошти і програмне забезпечення для управління мережею, а також процедури і процеси, які працюють разом.

Різні відділи компанії можуть використовувати свої апаратні пристрої в різних частинах однієї корпоративної мережі. Усіх їх об'єднує потреба спілкуватися один з одним, щоб працівники могли обмінюватися файлами, здійснювати телефонні дзвінки та виконувати всі інші важливі завдання, необхідні для ведення бізнесу.

Кожен пристрій використовує власну операційну систему та налаштування конфігурації, але вони повинні мати можливість спілкуватися один з одним за допомогою узгодженого протоколу, такого як TCP/IP або IPX/SPX. Щоб полегшити зв'язок між пристроями в корпоративній мережі, десь на лінії,

ймовірно, буде маршрутизатор або комутатор. Ці пристрої діють як дорожні поліцейські: вони отримують пакети від одного пристрою і перенаправляють їх до іншого на основі адрес, що містяться в цих пакетах.

Корпоративні мережі (рис. 1.1.) складаються з різних компонентів, які залежать від розміру компанії, її вимог та технологічних рішень. Часто до складу корпоративної мережі входять кінцеві пристрої, такі як ПК, ноутбуки, мобільні пристрої та сервери; мережеві пристрої, такі як репітори, мости, маршрутизатори, комутатори, брандмауери та сховища; комунікаційні протоколи; а також локальні мережі, такі як локальні мережі, глобальні мережі (WAN) та кампусні мережі (CAN).

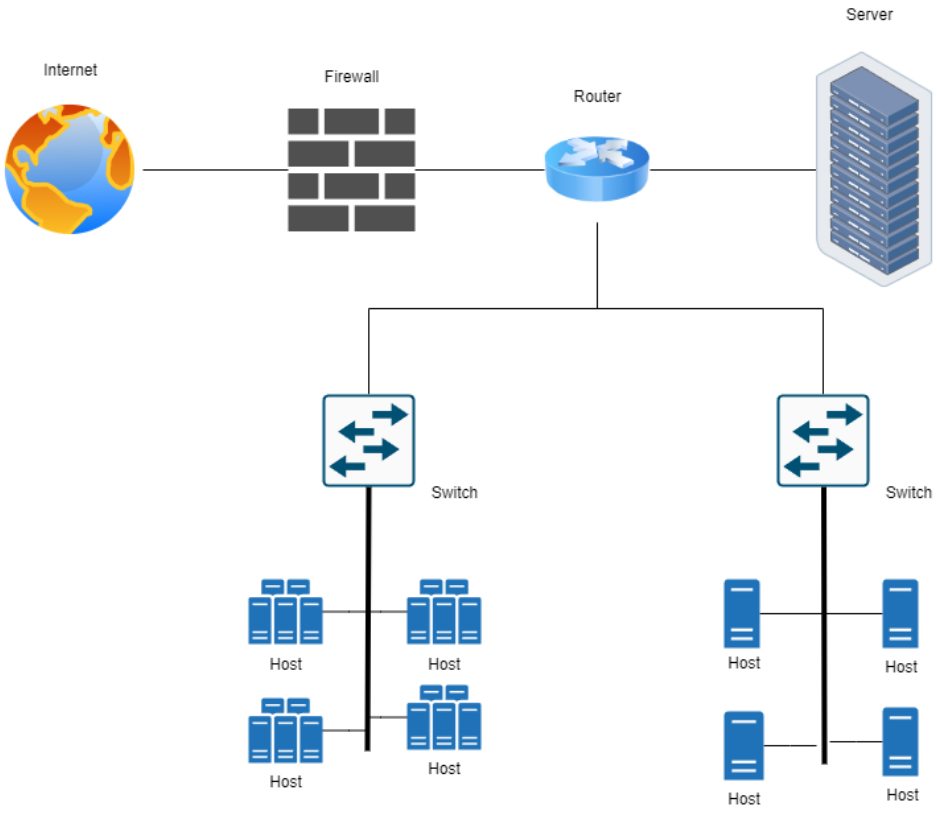


Рис. 1.1. Загальна схема корпоративної мережі

Маршрутизатори: Маршрутизація даних - фундаментальна функція граничних маршрутизаторів - передбачає переміщення даних через мережу від одного пристрою до іншого. Цей процес відбувається кожного разу, коли ми

користуємося інтернетом, від надсилання електронного листа до потокового перегляду фільму. Ось простий опис етапів, через які проходять маршрутизатори:

- Пакування: Коли ви хочете відправити інформацію через Інтернет (наприклад, надіслати електронний лист або отримати доступ до веб-сторінки), ця інформація розбивається на менші, керовані фрагменти, які називаються пакетами. Кожен пакет містить не лише частину загальних даних, але й адресу призначення - унікальний набір чисел - для того, куди він має надійти.
- Рішення про маршрутизацію: Кожен пакет надсилається на маршрутизатор, який зчитує адресу призначення. Маршрутизатор приймає рішення про те, куди відправити пакет далі, на основі інформації про мережу, яку він має.
- Стрибки: У більшості випадків пакети не надсилаються безпосередньо з пристрою-джерела на пристрій-приймач. Замість цього вони "перестрибують" через кілька маршрутизаторів на своєму шляху. Під час кожного переходу маршрутизатор дивиться на адресу призначення пакета і пересилає його найефективнішим маршрутом.
- Адресат досягнутий: Нарешті, пакет досягає місця призначення. Але це ще не все - пам'ятайте, що це лише невелика частина вихідних даних. Пристрій-одержувач повинен зібрати всі пакети і зібрати їх у вихідні дані.
- Підтвердження: Після того, як пристрій-одержувач отримав усі пакети і зібрав дані, він надсилає підтвердження назад на пристрій-джерело. Це дає змогу пристрою-джерелу знати, що дані були отримані правильно.
- Перевірка помилок і повторна передача: Якщо з якимось із пакетів виникла проблема (наприклад, якщо вони не надійшли або надійшли з помилками), пристрій-одержувач може попросити пристрій-джерело надіслати їх повторно. Це забезпечує цілісність даних.

Важливо зазначити, що кожен маршрутизатор у цьому процесі не знає повного шляху пакета. Він знає лише про мережі, які безпосередньо підключені до нього, і куди переслати пакет на основі найкращого маршруту, який він знає на той момент. Таблиці маршрутизації, які схожі на карту маршрутизатора, оновлюються динамічно на основі мережевих умов, саме тому інтернет вважається мережею з комутацією пакетів.

Маршрутизатори використовують різні протоколи, щоб дізнатися про найкращі шляхи, зокрема RIP (Routing Information Protocol), OSPF (Open Shortest Path First), BGP (Border Gateway Protocol) та інші. Ці протоколи маршрутизації дозволяють маршрутизаторам повідомляти про зміни в мережевих шляхах і гарантувати, що дані маршрутизуються якомога ефективніше.

Крім того, деякі маршрутизатори корпоративного класу тепер пропонують розширені функції. До них можна віднести програмно-визначені WAN-рішення (SD-WAN), які підвищують продуктивність і контроль трафіку додатків через резервні WAN-інтерфейси і VPN-з'єднання, приймаючи рішення про переадресацію на основі якості зв'язку. Ще одна вдосконалена функція - віртуальна маршрутизація і переадресація (VRF), яка дозволяє розділити один маршрутизатор на кілька окремих віртуальних доменів, що підвищує безпеку.

Комутатори(рис. 1.2): Працюють на рівні 2 (канальний рівень) моделі

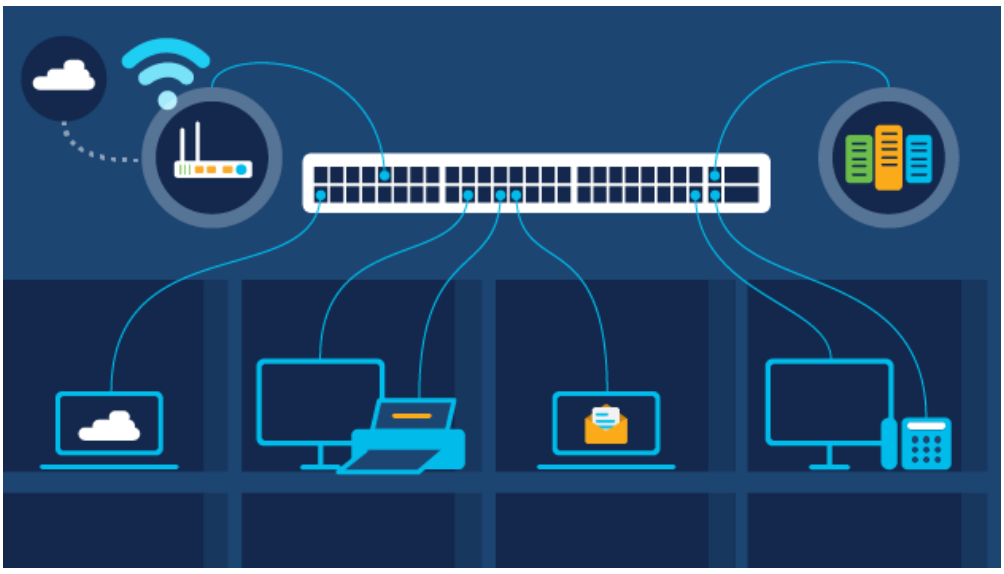


Рис. 1.2. Стандартна схема роботи комутатора

OSI, це мережеві апаратні пристрої, які використовуються для з'єднання декількох пристроїв в одній мережі; їх можна використовувати для управління фізичними мережами або програмними віртуальними пристроями. Після підключення пристрою до комутатора комутатор записує його MAC-адресу - код, який записаний на мережевій карті пристрою (NIC). NIC підключається до кабелю Ethernet, який з'єднує пристрій з комутатором. Комутатор використовує MAC-адресу, щоб визначити, з якого пристрою надсилаються вихідні пакети, і куди доставляти вхідні пакети.

MAC-адреса ідентифікує фізичний пристрій і не змінюється, в той час як IP-адреса мережевого рівня (рівень 3) може призначатися пристрою динамічно і змінюватися з часом. (Уявіть собі MAC-адресу як VIN-номер автомобіля, а IP-адресу - як номерний знак).

Коли пакет надходить на комутатор, він зчитує його заголовок, потім зіставляє з адресою або адресами призначення і відправляє пакет через відповідні порти, які ведуть до пристроїв призначення.

Щоб зменшити ймовірність колізій між мережевим трафіком, який одночасно надходить до комутатора і від нього, і підключеним пристроєм, більшість комутаторів підтримують повнодуплексний режим, в якому пакети, що надходять від пристрою і йдуть до нього, мають доступ до всієї смуги пропускання з'єднання комутатора. (Уявіть собі двох людей, які розмовляють по смартфону, а не по рації).

Хоча комутатори працюють на рівні 2, вони також можуть працювати на рівні 3, що необхідно для підтримки віртуальних локальних мереж (VLAN), логічних сегментів мережі, які можуть охоплювати підмережі. Для того, щоб трафік міг потрапити з однієї підмережі в іншу, він повинен пройти між комутаторами, і цьому сприяють можливості маршрутизації, вбудовані в комутатори.

Комутатори зазвичай з'єднують сегменти локальної мережі, тому концентратори підключаються до них. Комутатори відфільтровують трафік,

призначений для пристроїв у тому ж сегменті локальної мережі. Завдяки цій можливості комутатори ефективніше використовують власні обчислювальні ресурси, а також пропускну здатність мережі.

Різниця комутатора від маршрутизатора. Маршрутизатори обирають шляхи, якими пакети даних перетинають мережі та досягають пунктів призначення. Маршрутизатори роблять це, з'єднуючись з різними мережами і пересилаючи дані з мережі в мережу, включаючи локальні мережі, глобальні мережі (WAN) або автономні системи, які є великими мережами, що складають Інтернет.

На практиці це означає, що маршрутизатори необхідні для підключення до Інтернету, в той час як комутатори використовуються лише для з'єднання пристроїв між собою. Вдома і в невеликих офісах для доступу до Інтернету потрібні маршрутизатори, але більшість з них не потребують мережевих комутаторів, якщо тільки їм не потрібна велика кількість портів Ethernet. Однак великі офіси, мережі та центри обробки даних з десятками і сотнями комп'ютерів, як правило, потребують комутаторів.

Балансувальники навантаження (рис. 1.3): Балансувальники навантаження - це пристрої, які діють як "зворотний проксі" і розподіляють мережевий трафік між кількома серверами, щоб забезпечити ефективне використання ресурсів.

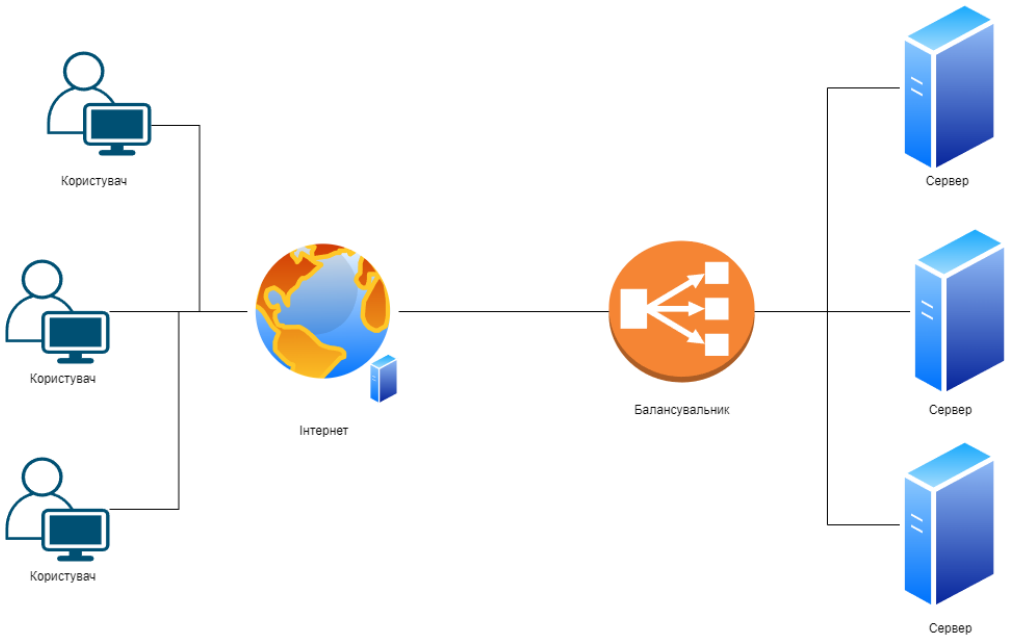


Рис. 1.3. Стандартне використання балансувальника навантаження

БН може бути як апаратним, так і програмним. Апаратні БН вимагають встановлення спеціального пристрою для балансування навантаження; БН навантаження можуть працювати на сервері, віртуальній машині або в хмарі. Мережі доставки контенту (CDN) часто включають функції БН. Коли від користувача надходить запит, БН призначає його певному серверу, і цей процес повторюється для кожного запиту. БН визначають, який сервер повинен обробляти кожен запит, на основі ряду різних алгоритмів. Ці алгоритми поділяються на дві основні категорії: статичні та динамічні.

Статичні алгоритми балансування навантаження розподіляють робочі навантаження без урахування поточного стану системи. Статичний БН не знає, які сервери працюють повільно, а які використовуються недостатньо. Замість цього він розподіляє навантаження на основі заздалегідь визначеного плану. Статичне БН швидко налаштовується, але може призвести до неефективності.

Повертаючись до наведеної вище аналогії, уявіть, що в продуктовому магазині з 8 відкритими касами є працівник, який має спрямовувати покупців до кас. Уявіть, що цей працівник просто йде по порядку, призначаючи першого покупця на лінію 1, другого - на лінію 2 і так далі, не озираючись, щоб побачити, як швидко рухаються черги. Якщо всі 8 касирів працюють ефективно, система працюватиме добре, але якщо один або декілька з них відстають, деякі черги можуть стати набагато довшими, ніж інші, що призведе до незадовільного обслуговування клієнтів. Статичне БН створює такий самий ризик: іноді окремі сервери можуть бути перевантажені.

Циклічний DNS і випадкове БН на стороні клієнта - дві поширені форми статичного БН.

Алгоритми динамічного БН враховують поточну доступність, завантаженість і стан кожного сервера. Вони можуть перенаправляти трафік з перевантажених або погано працюючих серверів на недостатньо завантажені, зберігаючи розподіл рівномірним і ефективним. Однак динамічне БН складніше

налаштувати. На доступність серверів впливає низка різних факторів: стан і загальна потужність кожного сервера, розмір завдань, що розподіляються, і так далі.

Припустимо, працівник продуктового магазину, який сортує покупців у чергах на касі, використовує більш динамічний підхід: він уважно стежить за чергами, бачить, які з них рухаються найшвидше, спостерігає, скільки продуктів купує кожен покупець, і відповідно до цього розподіляє покупців. Це може забезпечити більш ефективне обслуговування для всіх покупців, але це також створює більше навантаження на працівника, який сортує товари в черзі.

Існує кілька типів алгоритмів динамічного БН, зокрема найменше з'єднання, зважене найменше з'єднання, БН на основі ресурсів і геолокації.

БН часто використовується для веб-додатків. Програмні та хмарні БН допомагають рівномірно розподіляти інтернет-трафік між серверами, на яких розміщено додаток. Деякі хмарні продукти для БН можуть розподіляти навантаження інтернет-трафіку між серверами, розкиданими по всьому світу, - процес, відомий як глобальне БН на сервер (GSLB).

БН також широко використовується у великих локальних мережах, наприклад, у центрах обробки даних або великих офісних комплексах. Традиційно це вимагало використання апаратних пристроїв, таких як контролер доставки додатків (ADC) або спеціальний пристрій балансування навантаження. Для цієї мети також використовуються програмні БН.

Сервери: Сервер - це комп'ютерне обладнання або програмне забезпечення, яке надає послуги іншим користувачам мережі або пристроям, які називаються "клієнтами". Це комп'ютер, який постійно працює. Він завжди слухає запити і передає їх тому, хто зробив запит. Веб-браузер схожий на сервер. Ви можете зайти на веб-сайт, і якщо ви введете URL-адресу, наприклад, www.youtube.net, ваш веб-браузер звернеться до сервера за цією веб-адресою.

Сервер повідомить вашому веб-браузеру, куди відправити запит і яку веб-сторінку показати вам, коли він повернеться. Якщо ви хочете використовувати

один і той самий веб-браузер на кількох комп'ютерах, ви можете зберегти налаштування браузера, щоб не знати, куди звертатися.

Сервер є центром мережі, де зберігається вся інформація. Щоб отримати доступ до Інтернету, комп’ютер залежить від сервера. Тому дуже важливо, щоб сервер був надійним і мав достатньо оперативної пам’яті та жорсткого диска. Коли сервер знаходиться в мережі, він отримує дані від інших серверів і передає їх іншим.

Це полегшує взаємодію інших програм або пристроїв, які є клієнтами. Модель «клієнт-сервер» — це назва стилю архітектури, який включає кілька процедур або пристроїв в одному спільному обчисленні. Термін «служби» використовується для опису різноманітних можливостей, які мають сервери. Прикладами таких послуг є обмін інформацією та ресурсами між кількома клієнтами або виконання обчислень від імені клієнта.

Один сервер може обслуговувати декілька клієнтів одночасно. Клієнтська програма може виконуватися на одній машині. Ви можете використовувати сервер на іншому комп'ютері, і він буде доступний через мережу. Варіації на серверах включають:

- Сервери баз даних
- поштові сервери
- Серверне програмне забезпечення
- Файлові сервери
- Веб-сервери
- Ігрові сервери

Проксі-сервери (рис. 1.4.): Проксі-сервер виконує роль шлюзу між вами та інтернетом. Це сервер-посередник, що відокремлює кінцевих користувачів від веб-сайтів, які вони переглядають. Проксі-сервери надають різні рівні функціональності, безпеки та конфіденційності залежно від вашого випадку використання, потреб або політики компанії.

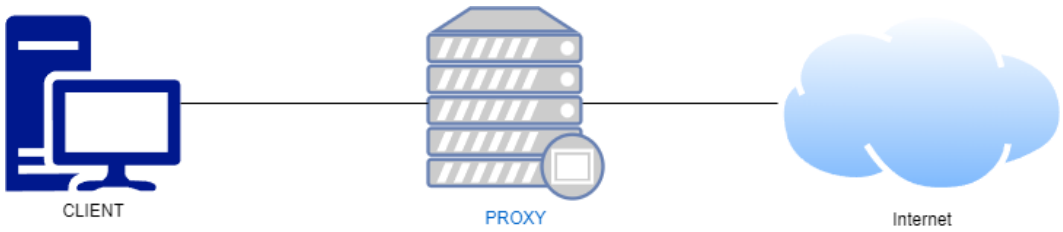


Рис. 1.4. Проксі сервер

Якщо ви використовуєте проксі-сервер, інтернет-трафік проходить через проксі-сервер на шляху до запитуваної вами адреси. Потім запит повертається назад через той самий проксі-сервер (існують винятки з цього правила), а потім проксі-сервер пересилає вам дані, отримані з веб-сайту.

Сучасні проксі-сервери роблять набагато більше, ніж просто пересилають веб-запити, і все це заради безпеки даних і продуктивності мережі. Проксі-сервери діють як брандмауер і веб-фільтр, забезпечують спільне мережеве з'єднання і кешують дані для прискорення загальних запитів. Хороший проксі-сервер захищає користувачів і внутрішню мережу від поганого, що “живе” в дикому інтернеті. І саме головне, проксі-сервери можуть забезпечити високий рівень конфіденційності.

Деякі з найпоширеніших типів проксі-серверів, які може використовувати компанія, включають наступні:

- Прямий проксі: Прямий проксі-сервер розгортається для захисту конфіденційності та безпеки користувачів. Весь інтернет-трафік від клієнтів всередині організації спрямовується через проксі-сервер.
- Зворотний проксі: Зворотний проксі призначений для захисту веб-сервера, а не кінцевого користувача. Трафік до захищених серверів з будь-якого джерела проходить через проксі-сервер, який може фільтрувати трафік і виконувати кешування.
- Анонімний проксі: Анонімні проксі-сервери приховують особистість клієнта, який запитує веб-сторінку. Спотворюючий проксі-сервер може не визнавати, що він є проксі-сервером, і може змінювати свою IP-адресу, щоб уникнути геолокації.

- Прозорий проксі: Прозорий проксі не приховує ніякої інформації про користувача. Він надсилає трафік на сервер, створюючи враження, що він надходить безпосередньо від користувача.
- Проксі центру обробки даних: Проксі центру обробки даних - це проксі, розгорнутий у центрі обробки даних. Усі запити користувачів проходять через центр обробки даних на шляху до місця призначення.

Порівняння проксі-серверу та VPN. І проксі-сервери, і віртуальні приватні мережі (VPN) маршрутизують трафік через сервер - або проксі-сервер, або кінцеву точку VPN - на шляху до місця призначення. Це дозволяє фільтрувати трафік, застосовувати політики та інші подібні переваги. VPN завжди шифрує трафік між користувачем і клієнтом, а трафік клієнта просто переадресується до місця призначення (тобто вихідною IP-адресою залишається IP-адреса клієнта). На відміну від цього, проксі-сервер може не шифрувати трафік на шляху до сервера і може запропонувати клієнту анонімність.

Проксі-сервери можуть надавати користувачам різні переваги. Однак вони також можуть мати свої недоліки, такі як:

- Вплив на продуктивність. Хоча проксі-сервер може підвищити продуктивність трафіку завдяки кешуванню, він може і зашкодити їй. Без оптимізованого апаратного забезпечення неефективна маршрутизація, пов'язана з проксі-серверами, може збільшити затримку в мережі.
- Журнали трафіку. Весь веб-трафік користувача проходить через проксі-сервер. Це може дозволити оператору сервера реєструвати цей трафік і використовувати або продавати ці журнали.
- Незашифрований трафік. Трафік до проксі-сервера може бути не захищений шифруванням. Це може дозволити підслуховуючому пристрою збирати інформацію про веб-трафік користувача.

Брандмауери: Це пристрої безпеки, які контролюють вхідний і вихідний мережевий трафік і приймають рішення про дозвіл або заборону доступу на

основі заздалегідь визначених правил безпеки. Спочатку брандмауери ділилися на два табори: проксі та з контролем стану. З часом перевірка стану стала більш складною, а продуктивність проксі-брандмауерів стала занадто повільною. Сьогодні майже всі брандмауери є брандмауерами з контролем стану і діляться на два основних типи: мережеві та комп'ютерні.

Брандмауери на основі хоста або комп'ютерні брандмауери захищають лише один комп'ютер, або "хост", і, як правило, встановлюються на домашніх або персональних пристроях, часто поставляються в комплекті з операційною системою. Іноді такі брандмауери також можуть використовуватися в корпоративному середовищі для забезпечення додаткового рівня захисту. Враховуючи той факт, що брандмауери на основі хостів необхідно встановлювати і підтримувати окремо на кожному пристрої, потенціал масштабування обмежений.

З іншого боку, міжмережеві екрани захищають всі пристрої і трафік, що проходять через точку розмежування, забезпечуючи широку масштабованість. Як випливає з назви, мережевий брандмауер функціонує на мережевому рівні, на 3 і 4 рівнях OSI, скануючи трафік між зовнішніми джерелами і вашою локальною мережею (LAN), або трафік, що рухається між різними сегментами всередині мережі. Вони розміщуються по периметру мережі або сегмента мережі як перша лінія захисту і контролюють трафік, виконуючи глибоку перевірку пакетів і фільтрацію пакетів. Якщо вміст пакетів не відповідає попередньо обраним критеріям на основі правил, створених мережевим адміністратором або командою безпеки, брандмауер відхиляє і блокує цей трафік.

У наступних розділах пояснюються відмінності між п'ятьма різними типами брандмауерів, згаданими вище, і те, що вони роблять для захисту мережі:

- Брандмауер з фільтрацією пакетів
- Брандмауер з перевіркою стану
- Шлюз канального рівня

- Шлюз на рівні додатків
- Брандмауер нового покоління (NGFW)

Брандмауер з фільтрацією пакетів -- Брандмауер з фільтрацією пакетів захищає мережу, аналізуючи трафік на рівні транспортного протоколу, де програми можуть взаємодіяти один з одним за допомогою певних протоколів: Протокол керування передачею (TCP) і Протокол користувацьких дейтаграм (UDP).

Брандмауер перевіряє пакети даних на цьому рівні, шукаючи шкідливий код, який може заразити мережу або пристрій. Якщо пакет даних ідентифікується як потенційна загроза, брандмауер відхиляє його. Малі підприємства, які потребують базового захисту від існуючих кіберзагроз, можуть скористатися брандмауером з фільтрацією пакетів.

Брандмауери з фільтрацією пакетів аналізують лише поверхневі дані і не відкривають пакет для перевірки фактичних даних (корисного навантаження). Вони перевіряють кожен пакет ізольовано на предмет призначення та IP-адреси, типу пакета, номера порту і мережевих протоколів, але не в контексті поточних потоків трафіку.

Брандмауер з перевіркою стану - Брандмауери з перевіркою стану працюють на шлюзі між системами за брандмауером і ресурсами за межами корпоративної мережі. Брандмауери з перевіркою стану знаходяться на 3 і 4 рівнях моделі взаємодії відкритих систем (OSI).

Брандмауери з перевіркою стану перевіряють кожен пакет (перевірка стану), а також відстежують і контролюють стан активних мережевих підключень, аналізуючи вхідний трафік на предмет потенційних ризиків. "Стан" - це найсвіжіший або поточний статус процесу або програми.

Брандмауери з підтримкою стану можуть виявляти спроби несанкціонованого доступу до мережі, а також аналізувати дані в пакетах на предмет наявності в них шкідливого коду. Вони дуже ефективно захищають мережу від атак типу "відмова в обслуговуванні" (DoS).

Важливо відстежувати стан і контекст мережевих комунікацій, оскільки ця інформація може бути використана для виявлення загроз - або на основі того, звідки вони надходять, або на основі того, куди вони йдуть, або на основі вмісту їхніх пакетів даних. Цей метод забезпечує більший рівень безпеки, ніж фільтрація пакетів або моніторинг каналів, але вимагає більшого навантаження на продуктивність мережі.

Шлюз канального рівня -- Шлюзи канального рівня працюють на рівні сеансу моделі взаємодії відкритих систем (OSI). У моделі OSI, перш ніж інформація може бути передана від одного кібер-суб'єкта до іншого, має відбутися рукостискання. Шлюзи канального рівня визначають безпеку встановленого з'єднання між транспортним і прикладним рівнями стека протоколів TCP/Інтернет (TCP/IP) шляхом моніторингу TCP-рукостискань між локальними і віддаленими хостами.

Хоча шлюзи канального рівня мають мінімальний вплив на продуктивність мережі, пакет даних, що містить шкідливе програмне забезпечення, може легко обійти шлюз канального рівня, навіть якщо він має легальне TCP-рукостискання. Це відбувається тому, що шлюзи канального рівня не фільтрують вміст пакетів даних. Щоб заповнити цю прогалину, шлюзи канального рівня часто використовують у парі з іншим типом брандмауера, який виконує фільтрацію вмісту.

Шлюз прикладного рівня, який також називають "проксі-брандмауером", слугує проміжною ланкою між внутрішніми та зовнішніми системами. Шлюз прикладного рівня працює на прикладному рівні, найвищому в моделі OSI. Він використовує глибоку перевірку пакетів (DPI) вхідного трафіку для перевірки як корисного навантаження (вмісту), так і заголовків пакетів даних. Цей брандмауер переконується, що на рівні додатків існують лише достовірні дані, перш ніж пропустити їх через себе.

Шлюзи рівня додатків дотримуються набору специфічних для додатків політик, щоб визначити, які з'єднання дозволені для проходження до і з додатків.

Вони допомагають захистити мережу, маскуючи запити клієнтів перед надсиланням їх на хост.

Коли потрібна мережева анонімність, часто використовують шлюзи на рівні додатків. Вони ідеально підходять для захисту веб-додатків від зловмисників (зловмисних намірів).

Брандмауер нового покоління (NGFW) - Брандмауер нового покоління (NGFW) - це єдиний тип брандмауера, який надає можливості для захисту сучасного бізнесу від нових кіберзагроз. Оскільки шкідливе програмне забезпечення і загрози стало складніше виявляти в точці доступу, безпека NGFW еволюціонувала, щоб охопити всю мережу і відстежувати поведінку і наміри.

NGFW надають такі функції, як глибока перевірка пакетів, запобігання вторгненням (IPS), розширене виявлення шкідливого програмного забезпечення, контроль додатків, а також забезпечують загальну видимість мережі завдяки перевірці зашифрованого трафіку. Вони можуть бути встановлені будь-де, від межі локальної мережі до її внутрішніх кордонів, а також можуть використовуватися в публічних або приватних хмарних мережах.

Процесоромісткі можливості NGFW включають розшифрування на дуже високому рівні продуктивності, глибоку перевірку пакетів після розшифрування, виявлення шкідливих URL-адрес, ідентифікацію командно-контрольної діяльності, а також завантаження шкідливого програмного забезпечення та кореляцію загроз. Завдяки цим розширеним можливостям безпеки, NGFW є критично важливими для суворо регульованих галузей, таких як фінанси або охорона здоров'я, і часто інтегруються з іншими системами безпеки та SIEM для наскрізної пильності та звітності.

На ринку існує широкий вибір NGFW з різними наборами функцій. Для того, щоб знайти найкращий варіант, організаціям було б корисно визначити функції безпеки, які їм найбільше потрібні в NGFW для їх галузі та сфери використання, щоб сфокусувати пошук.

Fortinet FortiGate NGFW поєднує в собі захист п'яти типів брандмауерів з розширеними можливостями безпеки, згаданими вище. Вони можуть бути розгорнуті як програмне або апаратне забезпечення і можуть масштабуватися в будь-якому місці: віддаленому офісі, філії, кампусі, центрі обробки даних і хмарі.

FortiGates - єдині NGFW з уніфікованим управлінням для гібридних міжмережевих екранів і послідовним забезпеченням безпеки в складних, гібридних середовищах.

Крім того, мережеві адміністратори також використовують програмне забезпечення для моніторингу та управління корпоративними мережами, оскільки ці інструменти дозволяють їм ефективно усувати проблеми, оптимізувати продуктивність і гарантувати, що мережа функціонує належним чином. До топу найкращих моніторингових програм увійшли:

- SolarWinds Network Performance Monitor. Набагато більше, ніж просто сканер. Більшість проблем з мережевою безпекою виникають при зміні конфігурації, а SolarWinds NPM виявляє зміни і може автоматично вирішити багато з них. Разом з надійним скануванням вразливостей і розширеними опціями для створення і моніторингу політик, це безумовно найкращий вибір для систем мережевого моніторингу.
- Auvik. Провідна система SaaS, заснована на мережевому картографічному засобі, що пропонує два рівні тарифних планів, які включають інструменти управління системою..
- Paessler PRTG Network Monitor. Програмне забезпечення для моніторингу мережі, яке використовує SNMP, сніфінг пакетів і WMI для моніторингу мережі.
- Datadog Network Monitoring. Забезпечує чудовий візуальний огляд компонентів вашої мережі та потоків мережевого трафіку між кожним компонентом. Цей підрозділ системних моніторів з хмарної

платформи забезпечує перевірку стану пристроїв і аналіз потоку трафіку.

- ManageEngine OpManager. Мережевий монітор, який може контролювати SNMP-пристрої, комутатори, сервери та віртуалізовані мережеві служби. Отримайте доступ до 30-денної безкоштовної пробної версії.
- Domotz. Ця SaaS-платформа пропонує широкий спектр послуг мережевого моніторингу від SNMP-опитування до сканування безпеки.
- Checkmk. Служба моніторингу всієї системи, а не лише мереж, доступна у безкоштовній та платній версіях. Працює на Linux або фізичному пристрої.
- NinjaOne. Цей пакет на основі RMM надає провайдерам керованих послуг інструменти для нагляду за мережами та кінцевими точками. Постачається з хмари.
- Site24x7 Network Monitoring. Комбінований сервіс моніторингу IT-інфраструктури, додатків та поведінки користувачів, доступний у хмарі.
- Intermapper від Fortra. Цей простий інструмент починається з інструменту автоматичного виявлення і мапує вашу мережу, а потім пропонує постійний моніторинг продуктивності.
- ExtraHop Reveal(x). Ця служба моніторингу безпеки мережі сканує загрози та підозрілу поведінку в режимі реального часу. Доступний у вигляді пакету SaaS або мережевого пристрою.
- Zabbix. Програмне забезпечення для моніторингу мережі з відкритим вихідним кодом з моніторингом SNMP та IPMP. Включає систему сповіщень та плагіни для спільноти.

- Catchpoint Network Experience. Цей модуль є частиною SaaS-пакету інструментів моніторингу повного стеку, які відстежують доставку веб-додатків.
- Nagios Core. Один з найкращих інструментів моніторингу мережі з відкритим вихідним кодом. Включає в себе інформаційну панель, систему сповіщень, плагіни для спільноти та багато іншого.
- Icinga. Система моніторингу мережі з відкритим вихідним кодом з DSL. Включає розширення.

Таблиця 1.1.

Порівняння сучасних систем моніторингу

Додаток/Особливості	SolarWinds NPM	Auvik	Paessler PRTG Network Monitor	Datadog	ManageEngine OpManager
SNMP Дані	+	+	+	+ Polls OIDs	+
Автоматичне виявлення	+ Рівень пристрою	+	+	+ Динамічне	+
Карта мережі	+	+	+ Інтерактивна	+	+ Користувацька
Сповіщення	+ За замовчуванням та користувацькі	+	+ Автоматизовані та сповіще	+	+ Можна налаштувати профайл сповіщень

			нння про помилки		
SaaS/On-Premises	On-premises	Saas	Saas, On-Premises	Saas	On-Premises
ОС	Windows Server	Cloud-based	Windows server	Cloud-based	Linux,Windws Server, Azure, and AWS
Безкоштовний тестовий період	30 діб	14 діб	30 діб	14 діб	30 діб

1.2 Технології та методи передачі даних в мережі: аналіз протоколів та механізмів обміну нформацією.

Передача даних в мережі є одним з найважливіших аспектів її роботи. Від того, наскільки ефективна і над надійна ця передача, залежить загальна продуктивність мережі і здатність її користувачів отримувати доступ до інформації та ресурсів.

Існує безліч різних технологій і методів передачі даних в мережі. Кожна з них має свої переваги і недоліки, і вибір оптимального рішення залежить від конкретних потреб мережі.

У цьому розділі ми розглянемо основні технології та методи передачі даних в мережі. Ми також проведемо порівняння різних підходів, щоб допомогти вам вибрати найкраще рішення для ваших потреб.

TCP/IP (Transmission Control Protocol/Internet Protocol) (рис. 1.5.).

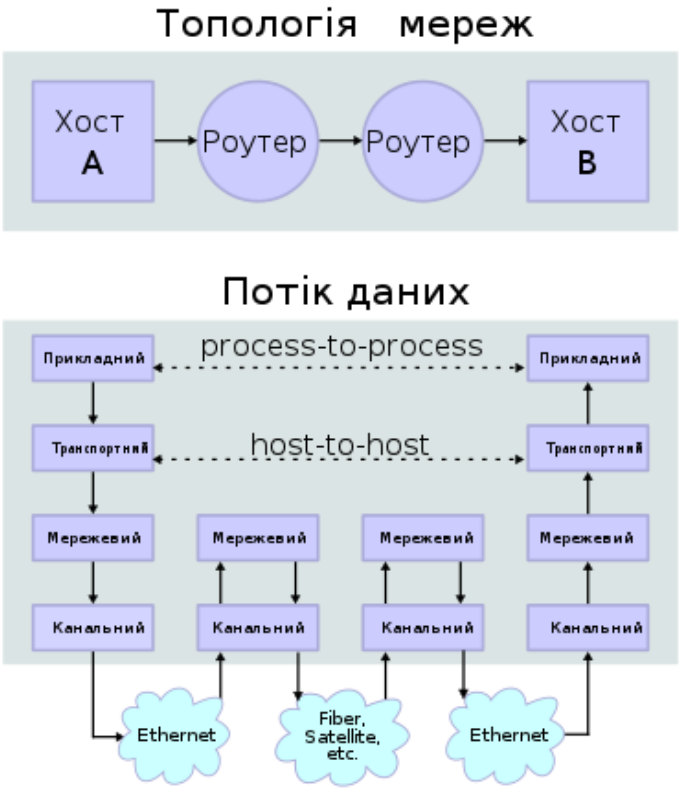


Рис. 1.5. TCP/IP модель

Модель TCP/IP - це метод передачі даних в Інтернеті за замовчуванням. Вона була розроблена Міністерством оборони США, щоб забезпечити точну і коректну передачу даних між пристроями. Вона розбиває повідомлення на пакети, щоб уникнути необхідності повторно надсилати все повідомлення, якщо воно зіткнеться з проблемою під час передачі. Пакети автоматично збираються знову, як тільки вони досягають місця призначення. Кожен пакет може пройти інший маршрут між комп'ютером-джерелом і комп'ютером-одержувачем, залежно від того, чи початковий маршрут, який використовувався, перевантажений або недоступний.

TCP/IP розподіляє завдання зв'язку на рівні, які забезпечують стандартизацію процесу, не вимагаючи від постачальників апаратного та програмного забезпечення самостійного управління. Пакети даних повинні пройти через чотири рівні, перш ніж вони будуть отримані пристроєм призначення, потім TCP/IP проходить через рівні у зворотному порядку, щоб повернути повідомлення до його початкового формату.

Як протокол, що базується на встановленні з'єднання, TCP встановлює і підтримує з'єднання між програмами або пристроями, поки вони не закінчать обмін даними. Він визначає, як початкове повідомлення має бути розбите на пакети, нумерує і збирає пакети, а потім надсилає їх до інших пристроїв у мережі, таких як маршрутизатори, шлюзи безпеки і комутатори, а потім до місця призначення. TCP також надсилає і приймає пакети з мережевого рівня, обробляє передачу будь-яких втрачених пакетів, керує потоком і гарантує, що всі пакети досягають місця призначення.

Гарним прикладом того, як це працює на практиці, є надсилання електронного листа за допомогою SMTP з поштового сервера. Щоб розпочати процес, рівень TCP на сервері ділить повідомлення на пакети, нумерує їх і пересилає на рівень IP, який потім транспортує кожен пакет до поштового сервера призначення. Коли пакети прибувають, вони повертаються на рівень TCP, де їх знову збирають у вихідний формат повідомлення і передають назад на поштовий сервер, який доставляє повідомлення до поштової скриньки користувача.

TCP/IP використовує тристороннє рукостискання для встановлення з'єднання між пристроєм і сервером, що забезпечує одночасну передачу даних через декілька TCP-сокетів в обох напрямках. Пристрій і сервер повинні синхронізувати і підтвердити пакети перед початком зв'язку, після чого вони можуть узгоджувати, роз'єднувати і передавати з'єднання TCP-сокетів.

Модель TCP/IP визначає, як пристрої повинні передавати дані між собою, і дозволяє здійснювати зв'язок через мережі та на великі відстані. Модель показує, як відбувається обмін даними та їхня організація в мережі. Вона складається з чотирьох рівнів, які встановлюють стандарти обміну даними і показують, як дані обробляються і пакуються під час передачі між додатками, пристроями і серверами.

Чотири рівні моделі TCP/IP є наступними (рис. 1.6.):

- Канальний рівень. Канальний рівень визначає, як мають надсилатися дані, керує фізичним процесом надсилання та отримання даних і

відповідає за передачу даних між додатками або пристроями в мережі. Сюди входить визначення того, як дані повинні сигналізуватися апаратним забезпеченням та іншими пристроями передачі в мережі, такими як драйвер комп'ютера, кабель Ethernet, мережева інтерфейсна карта (NIC) або бездротова мережа. Його також називають канальним рівнем, рівнем доступу до мережі, рівнем мережевого інтерфейсу або фізичним рівнем, і він є поєднанням фізичного та канального рівнів моделі взаємодії відкритих систем (OSI), яка стандартизує комунікаційні функції в обчислювальних і телекомунікаційних системах.

- Інтернет-рівень. Інтернет-рівень відповідає за відправлення пакетів з мережі та контроль їхнього переміщення мережею, щоб гарантувати, що вони досягнуть місця призначення. Він забезпечує функції та процедури для передачі послідовностей даних між додатками та пристроями в мережі.
- Транспортний рівень. Транспортний рівень відповідає за забезпечення надійного та надійного з'єднання між оригінальною програмою або пристроєм та місцем призначення. На цьому рівні дані розбиваються на пакети і нумеруються для створення послідовності. Потім транспортний рівень визначає, скільки даних потрібно відправити, куди і з якою швидкістю. Він гарантує, що пакети даних надсилаються без помилок і в послідовності, а також отримує підтвердження того, що пристрій призначення отримав пакети даних.
- Прикладний рівень. Прикладний рівень відноситься до програм, які потребують TCP/IP, щоб допомогти їм спілкуватися один з одним. Це рівень, з яким зазвичай взаємодіють користувачі, наприклад, системи електронної пошти та платформи обміну повідомленнями. Він

поєднує в собі сеансовий, представницький і прикладний рівні моделі OSI.

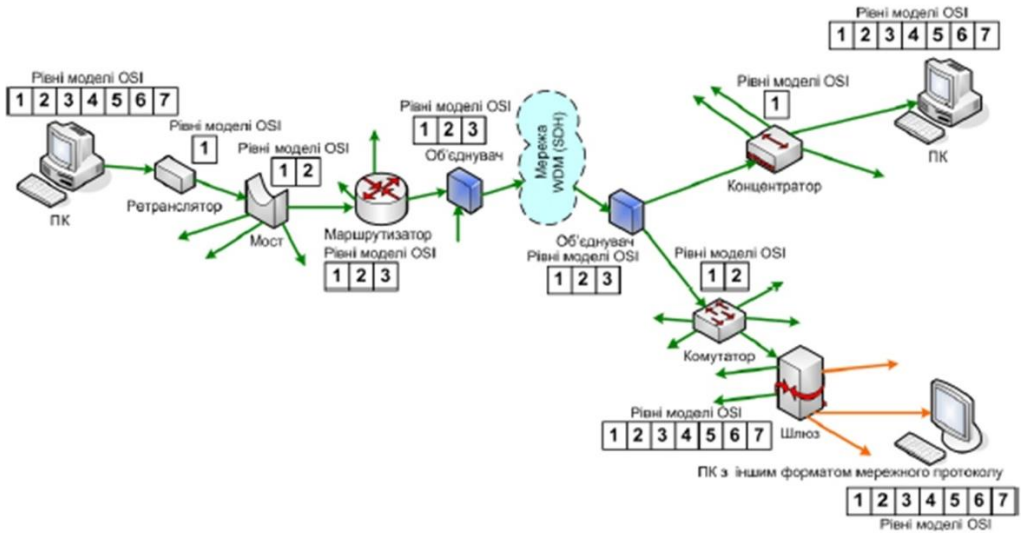


Рис. 1.6. Використання рівнів OSI у стандартній мережі

В сучасному світі інформаційних технологій, де сполучення комп'ютерів та мережевих пристроїв відіграє важливу роль, розуміння процесів передачі даних є необхідною основою для забезпечення ефективності та безпеки мережевого обміну інформацією. Одним із основних елементів цього процесу є використання IP пакетів та кадрів, що визначають способи організації та пересилання даних у мережі. У цьому контексті, розгляд аспектів формування та обробки IP пакетів, а також кадрів, є критичним для розуміння принципів роботи мережі та реалізації заходів її захисту. В даному вступному параграфі ми спробуємо глибше зануритися у світ передачі даних в мережі, де IP пакети та кадри виконують ключову роль у забезпеченні надійності та ефективності цього процесу. Кожен IP-пакет (рис. 1.7.) містить заголовок (довжиною 20 або 24 байти) і дані (змінної довжини). Заголовок містить IP-адреси джерела та призначення, а також інші поля, які допомагають визначити маршрут пакета. Дані - це фактичний вміст, наприклад, рядок літер або частина веб-сторінки.

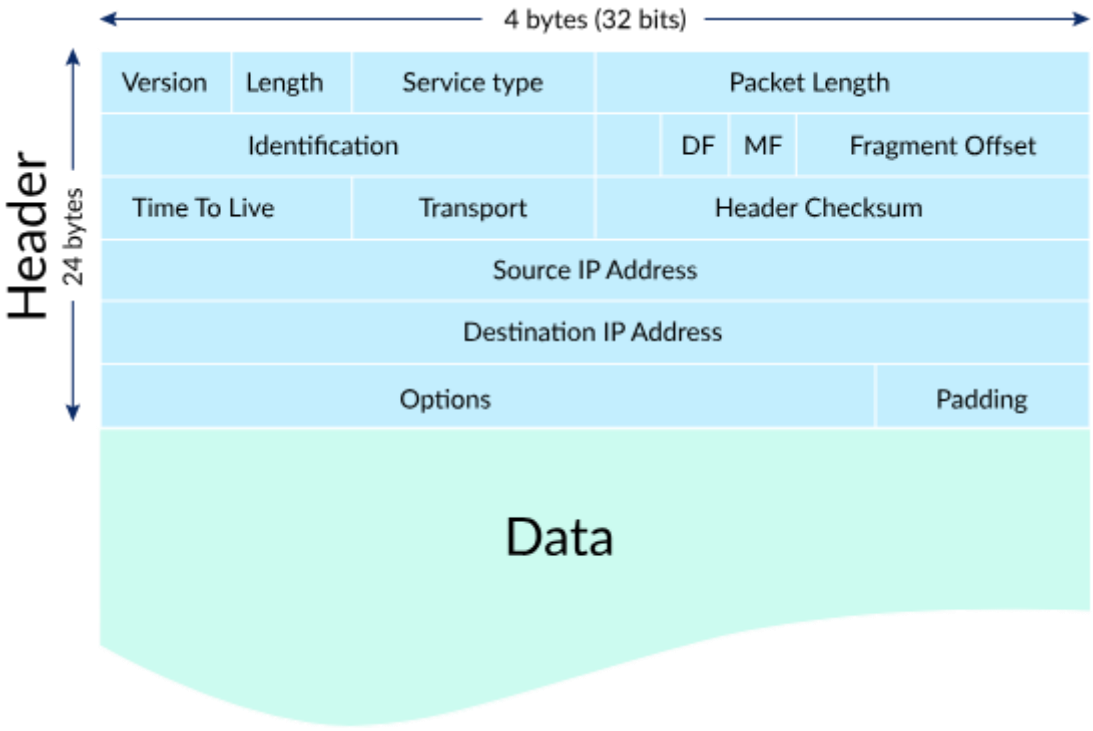


Рис. 1.7. Структура IP пакета

Передача даних на фізичному рівні передбачає синхронізовану передачу бітів від джерела до одержувача. На канальному рівні ці біти об'єднуються у кадри.

Зокрема, в комп'ютерних мережах і телекомунікаціях кадри є основними будівельними блоками цифрової передачі. Для передачі кадрів використовуються енергетичні пакети, які називаються фотонами, так само, як і світлова енергія. У процесі мультиплексування з часовим розділенням кадр використовується безперервно.

Кадри створюються на канальному рівні шляхом інкапсуляції пакетів, що надходять з мережевого рівня. Пакет може бути розділений на менші кадри, якщо розмір кадру стає занадто великим. Управління потоком і контроль помилок ефективніше контролювати за допомогою менших кадрів. Ці кадри передаються апаратним забезпеченням біт за бітом. Канальний рівень збирає сигнали від обладнання на стороні приймача і об'єднує їх у кадри. Кадр канального рівня складається з наступних компонентів(рис. 1.8.):

- Заголовок кадру містить керуючі байти, а також адреси джерела та одержувача кадру.

- Поле, яке містить інформацію, що має бути доставлена, називається корисним навантаженням (Payload).
- Причіп, який також називають послідовністю перевірки кадру (Frame Check Sequence, FCS), містить біти виявлення та виправлення помилок.
- Початок і кінець кадру позначаються двома прапорцями, розташованими на протилежних кінцях.

UDP (User Datagram Protocol). Протокол зв'язку для чутливих до часу додатків, таких як ігри, відтворення відео або пошук в системі доменних імен (DNS). UDP забезпечує швидший зв'язок, оскільки він не витрачає час на встановлення міцного з'єднання з місцем призначення перед передачею даних. Оскільки встановлення з'єднання займає певний час, усунення цього етапу призводить до збільшення швидкості передачі даних.

Однак UDP також може призвести до втрати пакетів даних на шляху від джерела до отримувача. Це також може полегшити хакеру виконання розподіленої атаки на відмову в обслуговуванні (DDoS).

У багатьох випадках, особливо за допомогою протоколу управління передачею (TCP), коли дані передаються через Інтернет, вони не тільки повинні бути відправлені з місця призначення, але й отримувач повинен сигналізувати, що він готовий до отримання даних. Як тільки обидва ці аспекти комунікації виконані, передача може розпочатися. Однак у випадку з UDP дані надсилаються до того, як з'єднання буде надійно встановлено. Це може призвести до проблем з передачею даних, а також дає можливість хакерам здійснювати DDoS-атаки.

У порівнянні з іншими мережевими протоколами, процес, що лежить в основі UDP, досить простий. Визначається цільовий комп'ютер і на нього надсилаються пакети даних, які називаються "дейтаграми". Немає нічого, що вказувало б на порядок, в якому пакети повинні надходити. Також не існує процесу перевірки того, чи дійшли дейтаграми до місця призначення.

Незважаючи на те, що UDP поставляється з контрольними сумами, які призначені для забезпечення цілісності даних, і номерами портів, які

допомагають диференціювати роль даних у джерелі та місці призначення, відсутність обов'язкового рукостискання створює проблему. Програма, яку користувач виконує за допомогою UDP, залишається вразливою до ненадійних аспектів базової мережі.

В результаті, дані можуть бути доставлені, а можуть і не бути. Крім того, порядок, в якому вони надходять, не контролюється, як це відбувається в TCP, тому те, як дані з'являються в кінцевому пункті призначення, може бути нестабільним, неправильним або містити порожні місця.

Однак у ситуації, коли немає необхідності перевіряти помилки або виправляти надіслані дані, це може не становити значної проблеми. Це одна з причин, чому UDP використовується у відео додатках. Вчасна доставка відеосигналу до місця призначення варта випадкових збоїв.

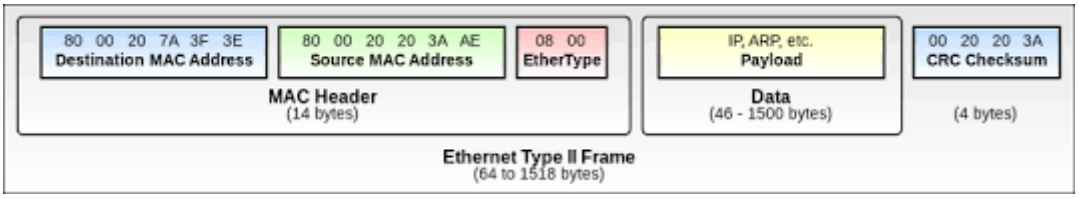


Рис. 1.8. Структура кадру канального рівня

HTTP/HTTPS (Hypertext Transfer Protocol/Secure). Кожне посилання, на яке ви переходите за допомогою HTTP, використовує базовий протокол, відомий як Hypertext Transfer Protocol (HTTP або "протокол"). HTTP - це стандарт мережевого протоколу, який визначає спосіб форматування і передачі повідомлень, а також дії веб-серверів і браузерів у відповідь на різні команди.

Щоразу, коли ви вводите URL-адресу у веб-браузері, ваш комп'ютер надсилає запит до сервера, на якому розміщений веб-сайт, який ви намагаєтеся відвідати. Цей сервер надсилає відповідь, зазвичай HTML-код веб-сайту. Цей зв'язок між вашим комп'ютером і сервером відбувається через порт 80 для незахищених з'єднань (тобто без використання протоколу SSL).

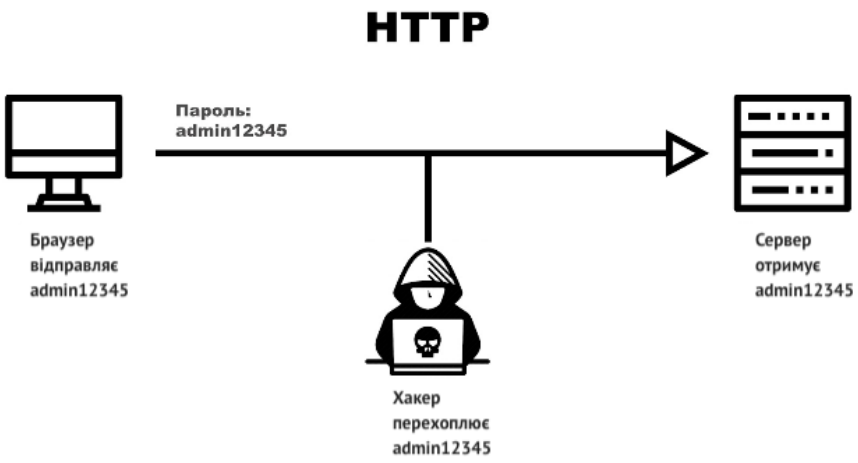


Рис. 1.9. HTTP з`єднання.

HTTPS (рис. 1.9.) - це розширення протоколу передачі гіпертексту (або просто "протокол"). Буква S в HTTPS означає "безпечний". Коли веб-сайт зашифрований за допомогою TLS (або SSL), він використовує захищений протокол передачі гіпертексту (HTTPS).

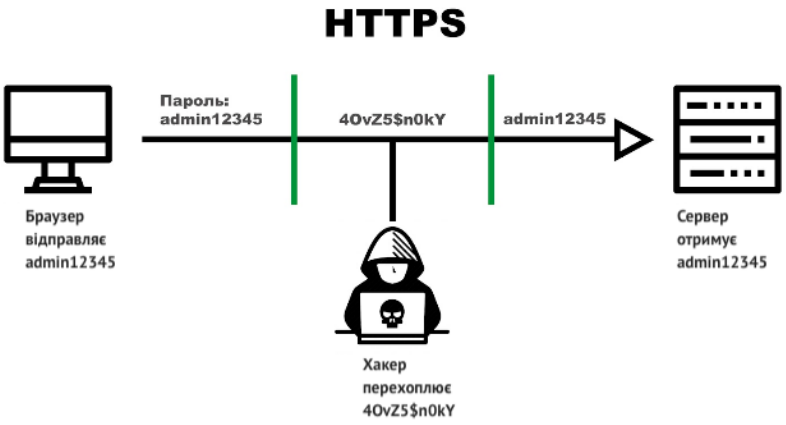


Рис. 1.10. HTTPS з`єднання.

По суті, це HTTP з шифруванням. Він використовується для захисту зв'язку через комп'ютерну мережу і широко застосовується в Інтернеті. HTTPS шифрує і розшифровує запити користувача на сторінки і сторінки, що повертаються веб-сервером.

Це захищає від атак типу "зловмисник посередині" і забезпечує конфіденційність даних, що передаються між браузером і веб-сайтом. За замовчуванням HTTPS-з'єднання використовують порт 443 (рис. 1.11.).

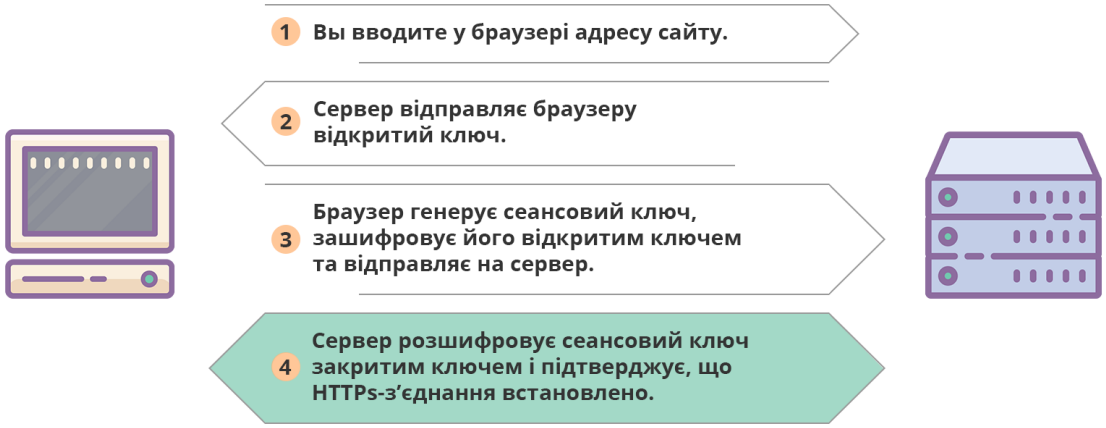


Рис. 1.11. Алгоритм встановлення https з`єднання

FTP (File Transfer Protocol). Є основним методом передачі файлів з 1970-х років. FTP - це простий спосіб переміщення файлів між комп'ютерами через TCP/IP - фреймворк, який з'єднує мережеві пристрої в Інтернеті. Ось як зазвичай працює FTP:

- Завантажте файли на FTP-сервер.
- Передайте ці файли через TCP/IP на FTP-хост.
- Одержувач отримує доступ до файлів і завантажує їх.

FTP працює з трьома форматами представлення даних (8-розрядні дані, ASCII 7-розрядні та EBCDIC 8-розрядні) і переміщує файли за допомогою одного з трьох режимів передачі (блоком, потоком і стиснутим).

Переваги FTP:

- Швидкий і простий, з перевіреною репутацією протягом чотирьох десятиліть.
- Можливість одночасної передачі декількох каталогів.

Недоліки FTP:

- Відсутність шифрування, що створює потенційні загрози безпеці.
- Використовує два канали передачі даних, що підвищує ризик несанкціонованого перехоплення даних.

FTPS (File Transfer Protocol Secure, іноді називають FTP/SSL) з'явився в 1990-х роках як посилена версія FTP. Він включає в себе рівень SSL/TLS, розташований під FTP, для шифрування каналів передачі даних, що підвищує безпеку під час передачі.

Переваги FTPS:

- Багато інтернет-інфраструктур мають вбудовану підтримку SSL, що спрощує передачу файлів через FTPS.
- Надійна автентифікація.
- Функції сертифіката X.509.

Недоліки FTPS:

- Може перешкоджати роботі брандмауерів, тому деякі користувачі можуть мати з ним проблеми на перших порах.

SFTP (Secure File Transfer Protocol - протокол захищеної передачі файлів), представлений наприкінці 1990-х років, пропонує зашифровану альтернативу FTP, що працює через SSH. Цей протокол забезпечує безпеку файлів під час передачі, що робить його надійним вибором для захисту даних від витоку.

Подібно до SSL, SFTP використовує команди для виконання з'єднання під час передачі файлів. Одержувач ваших файлів підключається до SSH-сервера і аутентифікує його за допомогою криптографічних ключів (SSH-ключів) або комбінації імені користувача і пароля.

Переваги SFTP:

- Шифрування файлів даних.
- Виконання команд.
- Підтримка IPV6 HTTP.
- Підтримка TMUX.
- Аутентифікація за допомогою імені користувача / пароля.
- Аутентифікація відкритим ключем.
- Один канал для передачі файлів.

- Відмінний вибір для різних типів плоских файлів, файлів з розділенням, звичайних текстових файлів, CSV-файлів, звичайних плоских файлів, файлів зі значеннями, розділеними комами, і файлів з простою структурою.

Недоліки SFTP:

- Недоліків небагато. SFTP є набагато безпечнішою альтернативою FTP, особливо для ETL.

Використовуючи протокол SSH, SCP (протокол безпечного копіювання) передає файли через зашифровані тунелі даних на основі IP, забезпечуючи швидкість і безпеку. Це відбувається шляхом переміщення файлів між локальними та віддаленими хостами (або двома віддаленими хостами).

Переваги SCP:

- Як і SFTP, SCP використовує протокол SSH для автентифікації, що робить його безпечнішою альтернативою FTP.
- Він (іноді) швидший за SFTP для передачі файлів, особливо у мережах з великою затримкою.

Недоліки SCP:

- Бракує можливостей керування файлами.
- Мало підтримує повторну передачу файлів.
- Створений лише для передачі файлів. На відміну від SFTP, ви не можете створювати каталоги, списки каталогів або видаляти файли. Залежно від конкретного типу даних, його можливості значно обмежені.

1.3 Дослідження сучасних алгоритмів та методів захисту віддалених підключень

В сучасному інформаційному середовищі важливість захисту віддалених підключень стала критичною, оскільки корпоративні та індивідуальні

користувачі все більше спираються на віддалені з'єднання для доступу до даних та ресурсів. Захист віддалених підключень стає визначальним у забезпеченні конфіденційності, цілісності та доступності інформації, що передається через мережі.

Однією з ключових причин важливості захисту віддалених підключень є збільшення обсягу роботи на віддалених робочих місцях та використання мобільних пристроїв. Забезпечення безпеки віддалених з'єднань стає важливим завданням для організацій, оскільки вони намагаються зберігати конфіденційні корпоративні дані та запобігати можливим атакам.

До інших аспектів важливості захисту віддалених підключень входить забезпечення відповідності стандартам безпеки та законодавчим вимогам. Компанії повинні враховувати вимоги щодо захисту особистих даних, особливо коли вони обробляють чутливу інформацію клієнтів чи користувачів.

Додатково, зростаюча кількість кіберзагроз та атак, спрямованих на віддалені мережі, підкреслює важливість захисту віддалених підключень. Використання захисту, такого як шифрування, ідентифікація та аутентифікація, допомагає уникнути несанкціонованого доступу та витоку конфіденційної інформації.

Загалом, у сучасному світі, де віддалена робота стає нормою, ефективні та надійні засоби захисту віддалених підключень визначають успіх та безпеку бізнесу та особистої інформації.

Безпека на транспортному рівні (Transport Layer Security, або TLS) - це широко розповсюджений протокол безпеки, розроблений для забезпечення конфіденційності та безпеки даних під час комунікацій через Інтернет. Основна сфера застосування TLS - шифрування зв'язку між веб-програмами та серверами, наприклад, веб-браузерами, що завантажують веб-сайт. TLS також можна використовувати для шифрування інших комунікацій, таких як електронна пошта, обмін повідомленнями та голосовий зв'язок через IP (VoIP). У цій статті ми зосередимося на ролі TLS в безпеці веб-додатків.

Протокол TLS був запропонований міжнародною організацією зі стандартизації Internet Engineering Task Force (IETF), а перша версія протоколу була опублікована в 1999 році. Остання версія - TLS 1.3, яка була опублікована в 2018 році.

TLS розвинувся з попереднього протоколу шифрування під назвою Secure Sockets Layer (SSL), який був розроблений компанією Netscape. TLS версії 1.0 фактично почав розроблятися як SSL версії 3.1, але назва протоколу була змінена перед публікацією, щоб вказати, що він більше не асоціюється з Netscape. Через цю історію терміни TLS і SSL іноді використовуються як взаємозамінні.

SSL розшифровується як Secure Sockets Layer (рівень захищених сокетів) і означає протокол для шифрування, захисту та автентифікації комунікацій, які відбуваються в Інтернеті. Хоча деякий час тому на зміну SSL прийшов оновлений протокол під назвою TLS (Transport Layer Security - безпека транспортного рівня), "SSL" все ще є загальновживаним терміном для позначення цієї технології.

Основна сфера застосування SSL/TLS - захист зв'язку між клієнтом і сервером, але вона також може захищати електронну пошту, VoIP та інші види зв'язку в незахищених мережах.

Це основні принципи, які необхідно засвоїти, щоб зрозуміти, як працює SSL/TLS:

- Безпечне з'єднання починається з рукостискання TLS, під час якого дві сторони, що спілкуються, відкривають захищене з'єднання та обмінюються відкритим ключем
- Під час TLS-рукостискання обидві сторони генерують сеансові ключі, а сеансові ключі шифрують і розшифровують всі повідомлення після TLS-рукостискання
- Для шифрування зв'язку в кожному новому сеансі використовуються різні сеансові ключі

- TLS гарантує, що сторона на стороні сервера або веб-сайт, з яким взаємодіє користувач, насправді є тим, за кого себе видає
- TLS також гарантує, що дані не були змінені, оскільки код автентифікації повідомлення (MAC) включається в передачу.

За допомогою TLS шифруються як HTTP-дані, які користувачі надсилають на веб-сайт (клікаючи, заповнюючи форми тощо), так і HTTP-дані, які веб-сайти надсилають користувачам. Зашифровані дані повинні бути розшифровані одержувачем за допомогою ключа.

Рукостискання TLS.Сеанси зв'язку TLS починаються з рукостискання TLS. Рукостискання TLS використовує так зване асиметричне шифрування, що означає, що на обох кінцях розмови використовуються два різні ключі. Це можливо завдяки техніці, яка називається криптографія з відкритим ключем.

У криптографії з відкритим ключем використовуються два ключі: відкритий ключ, який сервер робить загальнодоступним, і закритий ключ, який тримається в секреті і використовується тільки на стороні сервера. Дані, зашифровані за допомогою відкритого ключа, можуть бути розшифровані тільки за допомогою закритого ключа.

Під час рукостискання TLS клієнт і сервер використовують відкритий і закритий ключі для обміну випадково згенерованими даними, і ці випадкові дані використовуються для створення нових ключів для шифрування, які називаються сеансовими ключами.

Симетричне шифрування (рис. 1.12.) за допомогою сеансових ключів. На відміну від асиметричного шифрування, при симетричному шифруванні обидві сторони в розмові використовують один і той же ключ. Після рукостискання TLS обидві сторони використовують однакові сеансові ключі для шифрування. Після використання сеансових ключів публічні та приватні ключі більше не використовуються. Сеансові ключі - це тимчасові ключі, які більше не використовуються після завершення сеансу. Для наступного сеансу буде створено новий, випадковий набір сеансових ключів.

Рис. 1.12. Симетричне шифрування.

В сучасному цифровому вимірі, де доступ до інформації та сервісів стає все більше віддаленим та мобільним, питання забезпечення високого рівня безпеки та підтвердження ідентичності користувачів виявляється крайньою важливістю. Розділ "Аналіз протоколів аутентифікації та методів підтвердження ідентичності, включаючи двофакторну аутентифікацію (2FA)", присвячений вивченню та розгляданню різноманітних засобів, які використовуються для перевірки легітимності користувачів у цьому електронному ландшафті.

Відтворення надійного механізму впізнавання та аутентифікації стало завданням важливим для забезпечення конфіденційності та безпеки особистої та корпоративної інформації. Розглядаючи протоколи аутентифікації та методи підтвердження ідентичності, ми зосереджуємося на аналізі їхньої ефективності, безпеки та пристосованості до сучасних викликів кібербезпеки.

Особлива увага буде приділена двофакторній аутентифікації (2FA) як одному з перспективних методів підвищення рівня захисту. Враховуючи широкий спектр сценаріїв та технологій, які використовуються у процесі підтвердження ідентичності, ми дослідимо переваги та обмеження кожного методу.

У сучасному цифровому суспільстві, де наше життя нерозривно пов'язане зі світом, дуже важливо приділяти першочергову увагу захисту наших даних. Постійне зростання кіберзагроз і витоків даних підкреслює важливість впровадження методів автентифікації користувачів, які виходять за рамки паролів і є надійнішими за них. Двофакторна автентифікація (2FA), також відома як багатофакторна автентифікація (MFA), стала потужним щитом для захисту нашої цифрової сфери, гарантуючи, що лише уповноважені особи можуть отримати доступ до конфіденційної інформації.

По суті, двофакторна автентифікація - це процес, який вимагає від користувачів надання двох різних форм перевірки перед тим, як отримати доступ до своїх облікових записів. Перший фактор зазвичай включає щось відоме користувачеві, наприклад, пароль або PIN-код, що вже давно використовується для захисту облікових записів. Однак, покладаючись на паролі, акаунти можуть бути вразливими до атак грубої сили та методів підбору паролів.

Другий фактор двофакторної автентифікації вводить додатковий рівень безпеки. Це може бути щось, чим володіє користувач, наприклад, картка або фізичний токен безпеки, або щось, притаманне самому користувачеві, наприклад, біометричні дані. Біометрична автентифікація набула популярності завдяки своїй природі та стійкості до копіювання, адже кожна людина має свої унікальні біометричні дані, які можна використовувати для захисту власних даних у найкращий спосіб.

Біометрична автентифікація, така як сканування відбитків пальців, розпізнавання обличчя та райдужної оболонки ока, гарантує, що авторизовані користувачі можуть розблокувати цифрові двері акаунтів лише користувачем з цими біометричними даними. Хіба це не чудово?

Після введення імені користувача та пароля (перший фактор), система пропонує ввести другий фактор автентифікації. Цей додатковий крок перевірки може включати в себе одноразовий пароль (OTP), надісланий на мобільний пристрій користувача, який він повинен ввести, щоб продовжити роботу. Або ж це може бути сканування відбитків пальців, коли система порівнює відсканований відбиток з біометричними даними в базі даних.

Перевага двофакторної автентифікації полягає в її здатності протистояти кіберзагрозам, навіть якщо пароль користувача скомпрометований. Навіть якщо хакери дізнаються пароль, вони не зможуть отримати доступ до облікового запису без певної форми перевірки. Цей додатковий рівень безпеки значно знижує ризик доступу та витоку даних, що, звісно, є безцінним при захисті ваших даних.

Багато галузей та онлайн-платформ використовують двофакторну автентифікацію, щоб захистити своїх користувачів від потенційних порушень безпеки та кіберзлочинців, адже ми всі знаємо, наскільки вони спритні у своїх спробах вкрасти акаунти. Банківські установи, платформи соціальних мереж, поштові сервіси та веб-сайти електронної комерції - це лише кілька прикладів організацій, які впровадили цей надійний метод автентифікації. Насправді, багато сервісів роблять двофакторну автентифікацію обов'язковою для користувачів через її доведену ефективність у боротьбі з викраденням облікових даних і загальну безпеку.

Таким чином, двофакторна автентифікація слугує захистом наших ідентичностей і значно посилює автентифікацію користувачів, не покладаючись лише на паролі, адже, згідно з останніми дослідженнями, існують методи, за допомогою яких штучний інтелект зламує навіть найскладніші паролі менш ніж за годину. Звучить страшно, чи не так? Поєднуючи багатофакторну автентифікацію з такими методами, як риси та одноразові паролі, цей багаторівневий підхід запобігає кіберзагрозам і перешкоджає несанкціонованому входу. Використання переваг двофакторної автентифікації - це не тимчасова тенденція, а важливий крок до цифрового майбутнього, в якому ми зможемо впевнено і спокійно орієнтуватися у віртуальному світі. Тож давайте об'єднаємося, щоб прийняти це технологічне диво, і разом ми захистимо наші цифрові сфери, як ніколи раніше.

Існує кілька різних типів двофакторної автентифікації, тож давайте розглянемо їх ближче і зануримося в ці безцінні функції.

- 2FA за допомогою обладнання - це один з найстаріших типів 2FA. Він використовує апаратні токени, такі як брелок, який генерує цифровий код кожні 30 секунд, або може бути підключений до комп'ютера. Коли користувач намагається отримати доступ до свого облікового запису, він вводить відображений код підтвердження 2FA з фізичного пристрою в додаток або обліковий запис. Цей метод 2FA простий в реалізації і не

вимагає підключення до Інтернету. Оскільки він використовує апаратний токен, це один з найбезпечніших методів двофакторної автентифікації, хоча його налаштування та підтримка для кожного користувача може бути дорогим для бізнесу. Крім того, користувач може легко втратити або загубити апаратний пристрій, що може поставити вас у скрутне становище.

- SMS 2FA - цей метод верифікації просить користувача надати свій номер телефону. Коли користувач пізніше входить в систему, його просять ввести код підтвердження (зазвичай шість цифр), який надсилається на телефон користувача. Цей метод верифікації є популярним, оскільки більшість людей мають телефони з підтримкою SMS, і користувачеві не потрібно встановлювати додаток на свій телефон. Однак для отримання повідомлення користувачеві потрібен мобільний зв'язок, і якщо він втратить свою SIM-карту або телефон, він більше не зможе отримати доступ до повідомлення з підтвердженням автентичності. Зовсім недавно було виявлено недолік цього методу автентифікації, коли SMS-сервіси можуть дозволити хакерам заволодіти телефонними номерами за лічені хвилини, просто заплативши компанії за перенаправлення текстових повідомлень, і це може стати справжньою проблемою.
- 2FA за допомогою телефонного дзвінка - цей метод схожий на SMS 2FA, за винятком того, що користувач отримує телефонний дзвінок для отримання коду підтвердження. Цей метод верифікації має ті ж плюси і мінуси, що і метод SMS 2FA.
- 2FA через електронну пошту - 2FA через електронну пошту досить поширена, як і двофакторна автентифікація через SMS, коли користувач отримує електронний лист із секретним кодом або одноразовим паролем (OTP). У деяких випадках користувач може натиснути на унікальне посилання в електронному листі, щоб отримати доступ до

облікового запису замість пароля. Цей метод має ті ж переваги, що й SMS 2FA та 2FA за допомогою телефонного дзвінка, за винятком того, що для отримання електронного листа з підтвердженням потрібне підключення до Інтернету. Крім того, лист з підтвердженням може легко потрапити до папки "Спам" користувача, і, звичайно, якщо зловмисник має доступ до вашої електронної пошти, він отримає доступ і до онлайн-акаунта.

- Додаток-аутентифікатор / TOTP 2FA - цей метод перевірки вимагає від користувача завантаження додатку, наприклад, Google Authenticator, Microsoft Authenticator, Salesforce Authenticator або Authy. Коли користувач входить в онлайн-додаток з невідомого пристрою, він повинен відкрити додаток-автентифікатор, встановлений на його мобільному телефоні (або на комп'ютері, як у випадку з Authy). Додаток-автентифікатор генерує OTP-код - зазвичай шість-вісім цифр - який оновлюється кожні 30 секунд. Як тільки користувач вводить цей код у свій онлайн-акаунт, він отримує доступ. Однією з переваг є те, що додатки-автентифікатори легко впроваджувати і використовувати, коли користувач одразу отримує push-повідомлення з автоматично згенерованим кодом підтвердження, і йому не потрібно чекати на отримання SMS або електронного листа. З іншого боку, будь-хто, хто має доступ до телефону або комп'ютера користувача, може скомпрометувати ваш обліковий запис.
- 2FA через біометричні дані - цей тип 2FA є перспективною технологією, яка використовує біометричні дані користувача в якості токену - відбитки пальців, сітківку ока, а також розпізнавання обличчя або голосу. Цей метод верифікації є зручним для користувача, вважається найбезпечнішим типом 2FA і не вимагає підключення до Інтернету. Однак зберігання біометричних даних користувача може призвести до

порушення конфіденційності, і цей метод вимагає спеціальних камер і сканерів, але, звичайно, є найбезпечнішим з доступних.

- Резервні коди - Резервні коди є альтернативним методом верифікації, якщо користувач втратив свій мобільний телефон або не може отримати коди за допомогою SMS, голосового дзвінка або додатку для смартфона-автентифікатора. Якщо користувач не має свого ключа безпеки, він може використовувати ці одноразові коди для входу. Користувач може створити набір з 10 кодів у будь-який час. Після створення нового набору старий набір автоматично стає неактивним.

1.4 Висновки до розділу

Розділ "Архітектура та Засоби Захисту в Корпоративних Мережах" відкриває перед нами складні та важливі аспекти створення та управління мережевою інфраструктурою організації. Ми докладно розглянули основні компоненти корпоративної мережі, такі як маршрутизатори, комутатори, топології, та їхню роль у забезпеченні безпеки та надійності. Засоби захисту, такі як брандмауери, системи виявлення та запобігання вторгненням, антивірусне програмне забезпечення, виявилися ключовими елементами для забезпечення цілісності та конфіденційності корпоративної мережі.

У розділі "Технології та методи передачі даних в мережі: аналіз протоколів та механізмів обміну інформацією" ми пройшли від основних принципів передачі даних до деталей роботи різних протоколів, таких як TCP/IP, UDP, HTTP/HTTPS, та їхніх ролей у забезпеченні ефективного та безпечного обміну інформацією. Вивчення сучасних механізмів обміну даними надало нам можливість розуміти виклики, пов'язані з швидкістю передачі, безпекою та надійністю.

"Дослідження сучасних алгоритмів та методів захисту віддалених підключень" стало ключовим кроком у розумінні та застосуванні заходів для захисту даних у сучасному інформаційному середовищі. Ми розглянули різні методи аутентифікації, включаючи двофакторну аутентифікацію (2FA), а також технології, такі як VPN та SSH, що гарантують безпеку віддалених підключень. Важливість цього розділу виявилася у висвітленні необхідності боротьби з сучасними кіберзагрозами та забезпеченні захисту віддалених робочих середовищ.

У цілому, проведені дослідження дозволили визначити ключові аспекти та виклики, що стоять перед організаціями в сфері інформаційної безпеки та визначити оптимальні рішення для забезпечення безпеки, надійності та ефективності в корпоративних мережах та віддалених підключеннях.

РОЗДІЛ 2. АНАЛІЗ ІСНУЮЧИХ ЗАСОБІВ ЗАХИСТУ ВІДДАЛЕНИХ ПІДКЛЮЧЕНЬ.

2.1. Дослідження сучасного програмного забезпечення захисту віддалених підключень

Цей розділ присвячений дослідженню та розгляданню сучасних алгоритмів, які використовуються в VPN клієнтах для забезпечення безпеки та приватності під час віддаленого доступу до мережі. Вивчаючи ці алгоритми, ми розкриємо різноманітні аспекти та технічні рішення, які роблять сучасні VPN реалізації надійними та ефективними інструментами для захисту комунікацій. Найпоширеніші сучасні протоколи та алгоритми шифрування у VPN клієнтах:

- OpenVPN. Використовує AES-256 для шифрування даних
- WireGuard. Використовує ChaCha20 для шифрування даних

AES-256. Шифрування диктує культуру безпеки та конфіденційності в сучасному цифровому світі. Першим був стандартизований алгоритм Data Encryption Standard (DES). Однак, через його повільну обробку та легкість злому, він був замінений на більш надійний алгоритм AES.

AES розшифровується як Advanced Encryption Standard - це симетрична формула блочного шифрування з розміром блоку 128 біт. Це означає, що тільки відправник і одержувач мають ключ, щоб розшифрувати і зробити зашифрований текст нечитабельним для зловмисників. Це дозволяє перетворювати окремі блоки за допомогою ключів довжиною 128, 192 і 256 біт. Після того, як блоки зашифровані, алгоритм з'єднує їх разом, щоб сформувати зашифрований текст. AES базується на мережі підстановок і перестановок (SP-мережа), яка включає в себе заміну входів на певні виходи і перестановку.

Кожна довжина ключа має різну кількість можливих ключових комбінацій і варіюється в трьох варіантах:

- 128-бітна довжина ключа: 3.4 x 1038
- 192-бітний ключ: 6.2 x 1057
- 256-бітний ключ: 1.1 x 1077

Зрозуміти, як працює шифрування AES, досить просто. Один блок складається з 16 байт, що являє собою матрицю 4×4. Кожен байт має 8 біт, що в сумі створює блок довжиною 128 біт. Потім до кожного блоку застосовується алгоритм AES. Ключ, який використовується спочатку, розкладається на (n+1) ключів, де n - це кількість раундів у процесі шифрування.

Алгоритм виконує наступні шість кроків:

- Розширення ключа - створюються раундові ключі для використання на наступних етапах шифрування. Цей етап відповідає розкладу ключів.
- Додавання раунд ключа - раунд ключ є результатом виконання операції над попереднім раунд ключем і даними, які потрібно зашифрувати. Початковий ключ додається до розділених даних.
- Заміна байтів - Використовуючи підстановку S-box Rijndael, алгоритм AES замінює кожен байт кодом. На цьому етапі елементи в рядку блоку можуть виглядати наступним чином: 8d 08 05 5s
- Зсув рядків - рядки блоку від процесу заміни байтів зсуваються вліво. Перший рядок залишається, але другий зсувається вліво на один байт, третій - на два байти, а останній - на три байти.
- Перемішування стовпців - у попередньо визначеній матриці кожен стовпець перемножується, створюючи новий блок коду. Цей крок включає в себе багато складної математики.
- Додавання раунд ключа - на цьому кроці до стовпців додається раунд ключ, отриманий на кроці розширення ключа.

Цей процес повторюється для багатьох інших раундів, зокрема 9 разів для 128-бітового ключа, 11 разів для 192-бітового ключа і 13 разів для 256-бітового ключа.

ChaCha20-Poly1305. У світі кібербезпеки ChaCha20-Poly1305 - це алгоритм шифрування, який працює симетрично. Основна мета ChaCha20-Poly1305 - безпечно забезпечити ефективне шифрування та дешифрування даних. Він був створений Деніелом Бернштейном, відомим криптографом, відповідальним за більш відоме сімейство алгоритмів ChaCha. ChaCha20-Poly1305 працює як потоковий шифр, що означає, що він працює з окремими байтами даних. Якщо бути більш точним, він використовує 256-бітний ключ для шифрування даних. Крім того, він був розроблений для забезпечення швидкості та безпеки і вважається стійким до різних криптографічних атак, включаючи криптоаналіз і лінійний криптоаналіз. Завдяки своїй природі, ChaCha20-Poly1305 може бути легко реалізований в багатоядерних процесорах і різних високопродуктивних системах.

Основні етапи процесу шифрування ChaCha20-Poly1305:

- Генерація ключа: Алгоритм ChaCha20-Poly1305 генерує 256-бітний ключ з ключа, наданого користувачем. Після цього випадковим чином генерується 96-бітний nonce.
- Процес ініціалізації: У цьому стані алгоритм ChaCha20-Poly1305ініціалізує стан шифру за допомогою ключа і nonce.
- Блокове шифрування: Використовуючи стан шифру, алгоритм шифрує кожен блок даних, а потім стан шифру оновлюється після шифрування кожного блоку.
- На виході: Нарешті, зашифрований текст або зашифрована інформація створюється за допомогою XORing.

ChaCha [NL15] оперує з 32-бітними словами, отримує на вхід 256-бітний ключ K = $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7)$ та 32-бітний лічильник C = (c0), і видає

послідовність 512-бітних блоків ключового потоку1. Ця функція діє на матрицю 4 × 4 з 32-бітних слів, записаних у вигляді:

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} = \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ c_0 & n_0 & n_1 & n_2 \end{pmatrix}$$

σ і τ - константи, де

$$(\tau_0, \tau_1, \tau_2, \tau_3) = (0 \times 61707865, 0 \times 3120646E, 0 \times 79622D36, 0 \times 6B206574)$$

$$(\sigma_0, \sigma_1, \sigma_2, \sigma_3) = (0 \times 61707865, 0 \times 3320646E, 0 \times 79622D32, 0 \times 6B206574).$$

Блок клавішного потоку Z визначається наступним чином; Z = X + X(20), де X(r) = Roundr(X) з круглою функцією ChaCha і + є послівним додаванням за модулем 232. Якщо Z = X + X(r) то це називається "r-раунд ChaCha" або "ChaChar". Функція раунд додавання складається з наступних нелінійних операцій, які називаються 4х раундові функціїї. Вектор (a, b, c, d) з чотирьох слів перетворюється як:

a = a + b

d = d ⊕ a

d = (d)≪16

c = c + d

b = b ⊕ c

b = (b)≪12

a = a + b

d = d ⊕ a

d = (d)≪8

c = c + d

b = b ⊕ c

b = (b)≪7

4х раундові функціїї застосовуються до стовпчиків(x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$ та $(x_{15}, x_3, x_7, x_{11})$ у непарному раунді та діагоналях $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}) та (x_3, x_4, x_9, x_{14}) у парних раундах. Алгоритм описує повну процедуру ChaCha

Псевдокод алгоритму (рис. 2.1.):

Input: Key K, Counter C, and Nonce N
Output: Keystream Z
Generate initial matrix X using K, C, and N
$y \leftarrow X$
for $i \leftarrow 0$ **to** 9 **do**
/* Column Round */
$(x_0, x_4, x_8, x_{12}) \leftarrow quarterround(x_0, x_4, x_8, x_{12})$
$(x_5, x_9, x_{13}, x_1) \leftarrow quarterround(x_5, x_9, x_{13}, x_1)$
$(x_{10}, x_{14}, x_2, x_6) \leftarrow quarterround(x_{10}, x_{14}, x_2, x_6)$
$(x_{15}, x_3, x_7, x_{11}) \leftarrow quarterround(x_{15}, x_3, x_7, x_{11})$
/* Diagonal Round */
$(x_0, x_5, x_{10}, x_{15}) \leftarrow quarterround(x_0, x_5, x_{10}, x_{15})$
$(x_1, x_6, x_{11}, x_{12}) \leftarrow quarterround(x_1, x_6, x_{11}, x_{12})$
$(x_2, x_7, x_8, x_{13}) \leftarrow quarterround(x_2, x_7, x_8, x_{13})$
$(x_3, x_4, x_9, x_{14}) \leftarrow quarterround(x_3, x_4, x_9, x_{14})$
end for
$Z \leftarrow X + y$
return Z

Рис. 2.1. Псевдокод алгоритму ChaCha20

У таблиці нижче ви можете побачити основні відмінності між двома типами шифрування:

Таблиця 2.1

Порівняння AES та ChaCha20 алгоритму шифрування інформації

AES	**ChaCha20**
128, 192 or 256 біт	256 біт
Блоковий шифр	Потоковий шифр
Застарілий крипто-алгоритм	Новий крипто-алгоритм
Комплексний	Простий
Схильність до людських помилок	Не схильний
Потрібне обладнання	Працює без додаткового обладнання
Може бути повільним при шифрувані	Працює стабільно швидко

Для простоти ми порівняємо 256-бітові версії обох алгоритмів шифрування (таб 2.1). Довгі ключі шифрування, без сумніву, роблять обидва ці набори шифрування дуже стійкими. Вони обидва кращі з точки зору безпеки, ніж будь-

який з їхніх аналогів, що використовують коротші ключі шифрування. Однак, шифрування - це набагато більше, ніж довжина ключа.

Основна відмінність між шифруванням AES-256 і XChaCha20 полягає в тому, що AES-256 - це блоковий шифр, тоді як XChaCha20 - потоковий шифр. Крім того, шифрування AES має досить хорошу репутацію (тому його називають "просунутим стандартом шифрування"), тоді як XChaCha20 все ще є досить новим.

Зробимо порівняння швидкості цих двох алгоритмів. З реалізацією постійного часу з бітовою розбивкою на 32-розрядні регістри; два блоки обробляються паралельно, коли це дозволяє режим шифрування (розшифровка CTR і CBC, але не шифрування CBC).

Таблиця 2.2

Порівняння середньої швидкості шифрування/дишифрування AES-256 та ChaCha20

Алгоритм	**amd64 (MB/s)**	**i386 (MB/s)**	**pwr8 (MB/s)**	**m0+ (kB/s)**
ChaCha20	322.54	270.72	259.51	550.72
AES-256	40.07	31.58	30.16	82.70

Як бачимо біль сучасніший алгоритм ChaCha20 більш продуктивний за AES-256. Також алгоритм ChaCha20 не залежить від апаратної частини, так як алгоритм AES-256, що дає виграш працюючи на мобільних девайсах та інших пристроях ARM архітекутри.

Сучасні VPN протоколи, які використовують вище наведині алгоритми. Включають в себе:

Перше, що потрібно зрозуміти про OpenVPN - це те, що це VPN-протокол. Протокол - це набір правил, який створює VPN-тунелі. Він визначає, як шифрувати і маркувати пакети даних, що проходять через VPN-з'єднання.

Існує багато різних варіантів протоколів для користувачів VPN, включаючи L2TP, TLS/SSL і IPSec. Кожен з них має свої сильні сторони, але OpenVPN вважається одним з лідерів галузі.

OpenVPN з'явився у 2001 році і був першим VPN-протоколом з відкритим вихідним кодом. З тих пір з'явилася глобальна спільнота розробників. Спільнота OpenVPN з відкритим вихідним кодом перевіряє помилки і вдосконалює протокол, додаючи нові функції і оновлюючи елементи безпеки. Постійний контроль означає, що протокол рідко залишається поза увагою нових кіберзагроз.

OpenVPN є частиною сімейства протоколів VPN із захищеним тунелюванням сокетів (SSL). Це означає, що він працює на транспортному рівні (рівень 4) моделі OSI. Це також надає OpenVPN доступ до величезної бібліотеки SSL, що забезпечує безліч можливостей налаштування при створенні VPN.

Протокол OpenVPN використовує обмін ключами SSL/TLS для шифрування при створенні VPN-тунелів. Цей процес стандартно використовує 256-бітове шифрування - рівень захисту, який не може зламати навіть АНБ.

Якщо 256-бітного шифрування недостатньо, OpenVPN може піти ще далі. Протокол підтримує передові методи шифрування, такі як Blowfish, AES і CAST-128, що робить передачу даних майже невразливою для зовнішніх атак.

Крім того, OpenVPN використовує Perfect Forward Secrecy (PFS). PFS створює унікальний ключ шифрування для кожного сеансу або передачі даних. Заміна ключів шифрування ускладнює для зовнішніх зловмисників викрадення ключів та обхід шифрів шифрування.

Окрім шифрування, OpenVPN також використовує розширені процедури автентифікації. Автентифікація гарантує, що кожен пакет даних доставляється на правильну адресу і в правильній послідовності.

Коли користувачі надсилають дані через OpenVPN, інструмент під назвою TLS-аутентифікація застосовує автентифікацію за допомогою хеш-коду HMAC (Hash Message Authentication Code). HMAC практично гарантує точну передачу даних з мінімальною втратою даних.

OpenVPN використовує два режими передачі: UDP і TCP. Ми розглянемо обидва пізніше. Але зараз важливо відзначити, що тільки режим TCP надає

послуги автентифікації. TCP перевіряє передачу, щоб відстежувати втрачені пакети, в той час як UDP може призвести до втрати даних.

Пакети OpenVPN можуть проходити через будь-який порт. Використання протоколів SSL означає, що трафік, який передається через мережу, практично неможливо відрізнити від звичайних даних. При правильному налаштуванні порту 443 сторонні особи навіть не зможуть виявити використання VPN.

Ця конфігурація порту також дозволяє OpenVPN дуже добре обходити брандмауери - проблема, яка може бути характерною для IPSec VPN.

OpenVPN став основним протоколом для багатьох комерційних віртуальних приватних мереж, і на це є багато вагомих причин. Переваги протоколу включають в себе

- Надійне шифрування з великою кількістю опцій. 256-бітове шифрування є стандартним режимом протоколу OpenVPN і має захистити дані від усіх кібер-зловмисників. Налаштування за замовчуванням забезпечує безпеку конфіденційних даних, таких як платіжні реквізити або логіни компанії. А якщо користувачі хочуть більш надійного шифрування, за потреби можна додати інші шифри.
- Легко масштабується. Системи OpenVPN можуть обслуговувати одну робочу станцію, підключену до мережі компанії, або масштабуватися до систем безпеки всього підприємства. VPN розроблена таким чином, щоб охоплювати стільки пристроїв, скільки потрібно користувачам. Користувачі завантажують попередньо налаштовані клієнти, встановлюють їх на свої пристрої і готові до роботи.
- Відмінна сумісність пристроїв. Однією з найбільших переваг OpenVPN є те, що користувачі можуть встановлювати клієнти OpenVPN на Windows, Linux і macOS. Протокол пропонує функціональність для Android та iOS і може бути налаштований для конкретного обладнання, якщо це необхідно.

- Продуктивність брандмауера. Не всі VPN можуть легко проходити через NAT-шлюзи або брандмауери. Це не проблема для OpenVPN, який відомий своєю здатністю працювати з брандмауерами та іншим фільтруючим обладнанням.
- Гнучкість протоколу. Можливість перемикання між UDP і TCP - ще одна сильна сторона протоколів OpenVPN. Геймери та стримери можуть насолоджуватися швидкістю передачі UDP. TCP доступний для передачі даних з високим рівнем безпеки, коли швидкість менш критична.
- Дружній до плагінів. OpenVPN працює зі сторонніми надбудовами та плагінами. Це розширює можливості базової VPN, додаючи різноманітні сервіси. Ці послуги включають гнучкі варіанти автентифікації - зручна функція для корпоративних мереж. Плагіни також включають безліч інструментів для створення серверів OpenVPN.
- Поширення з відкритим вихідним кодом. Більшість протоколів VPN належать корпораціям. Наприклад, Microsoft і Cisco володіють L2TP. Але OpenVPN залишається з відкритим вихідним кодом, що забезпечує прозорість кодової бази. Користувачам не потрібно покладатися на те, що корпорації захистять дані та уникнуть помилок безпеки. Світова спільнота розробників допрацьовує OpenVPN відповідно до останніх тенденцій у сфері кібербезпеки.

Перераховані вище переваги відрізняють OpenVPN від більшості інших VPN-протоколів. Але жодна технологія кібербезпеки не є бездоганною. OpenVPN не є винятком, і у нього є кілька потенційних недоліків, про які користувачі повинні знати:

- Не завжди найшвидший варіант. У світі VPN безпека має свою ціну. Це стосується і OpenVPN, особливо при використанні захищеного режиму передачі UDP. При використанні надійної автентифікації та шифрування

OpenVPN працює приблизно так само швидко, як L2TP, хоча інші протоколи тунелювання можуть бути швидшими.

- Може бути складним у налаштуванні. Користувачам може бути складно налаштувати систему OpenVPN з нуля. Засновники VPN рекомендують 20-етапний процес ручного налаштування, і кожен крок є складним і трудомістким. На практиці більшість людей купують готовий OpenVPN-клієнт з потрібними їм функціями. Це зручно, але компанії з індивідуальними потребами в безпеці можуть знайти інші протоколи більш зручними для користувачів.
- Не найкращий VPN для мобільних пристроїв. OpenVPN став спрощеним рішенням для операційних систем настільних комп'ютерів і ноутбуків, але не настільки зручним для мобільних користувачів. Реалізації для Android та iOS не такі просунуті та зручні для користувачів, хоча вони постійно вдосконалюються.

WireGuard. Це мережевий протокол для VPN, який пропонує кращий захист і продуктивність, ніж інші технології тунелювання. На відміну від IKEv2, IPSec і OpenVPN, які працюють у режимі користувача, код WireGuard працює в ядрі разом з іншими файлами операційної системи.

Ще кілька переваг WireGuard:

- Низьке використання ресурсів. Протокол використовує принцип асиметричної криптографії, тому дані шифруються без ресурсоємних операцій на сервері. Це значно знижує навантаження. WireGuard підходить для встановлення на малопотужних машинах, таких як домашні сервери для IoT-пристроїв або маршрутизатори.
- Продуктивність. WireGuard працює швидше, ніж OpenVPN та IPsec, завдяки мінімалістичній архітектурі та оптимізації коду. При підключенні до порту 1 Гбіт/с пропускна здатність WireGuard досягає 1011 Мбіт/с, що майже в 4 рази швидше, ніж у OpenVPN.

- Низька затримка. Пінг з WireGuard в 3,5 рази нижче, ніж з OpenVPN. Це також пов'язано з роботою на стороні ядра: протоколу не потрібно копіювати дані з середовища користувацького режиму і назад, як це робить більшість VPN, щоб обробити код. WireGuard також має заздалегідь визначену конфігурацію з'єднання між сервером і клієнтом, що скорочує час з'єднання.
- Високий рівень безпеки. Протокол використовує асиметричні ключі Curve25519 і Poly1305, які забезпечують кращий захист, ніж RSA і AES, що використовуються в інших VPN. Ключі доступу генеруються автоматично і тільки для 2 конкретних пристроїв, що виключає їх централізоване зберігання і підвищує безпеку протоколу.
- Програмний код WireGuard складається всього з 4 000 рядків, що полегшує його інтеграцію. Крім того, протокол включений в ядро Linux версії 5.6 і вище, що забезпечує високу сумісність з усіма операційними системами сімейства. Протокол також надає гнучку документацію API для створення VPN конфігурацій: WireGuard набагато простіше підключати і налаштовувати, ніж OpenVPN і IPsec.

Однак, щоб отримати щось велике, ми повинні чимось пожертвувати натомість. Тому WireGuard також має деякі недоліки:

- Вбудована підтримка. Хоча WireGuard пропонує клієнтські програми для всіх основних платформ, він не працює без додаткового програмного забезпечення, за винятком деяких дистрибутивів Linux. Якщо ви хочете використовувати VPN на пристрої, на який не можна встановити програми, вам доведеться використовувати інший протокол.
- Заплутаність. Проект WireGuard не ставить за мету створення VPN, яка протидіє глибокому інспектуванню пакетів. Якщо, наприклад, ви намагаєтеся пройти через Великий китайський брандмауер, WireGuard сам по собі не впорається з цим завданням. Однак архітектура WireGuard дозволяє йому підтримувати тунелі обфускації в якості додаткового шару.

Порівняння швидкості OpenVpn та WireGuard алгоритмів. Результати тестування швидкості WireGuard проти OpenVPN

- WireGuard був найшвидшим у 58,8% тестів на завантаження.
- В середньому WireGuard був на 14,6% швидшим за OpenVPN на UDP і на 56,1% швидшим за OpenVPN на TCP.
- Середня втрата швидкості склала 19,1% для WireGuard, 20,6% для OpenVPN на UDP і 58,1% для OpenVPN на TCP.

Таблиця 2.3.

Порівняння протоколів VPN

Протокол	Алгоритм шифрування	Аутентифікація	Використання
OpenVPN	AES-256, Blowfish, Camellia, also supports ChaCha20	Підтримує Poly1305	Повсякденне використання; встановлення VPN на роутери
IKEv2/IPSEC	AES-256	Підтримує HMAC та AES	Мобільні пристрої, підключення ближньої дії, повсякденне використання
WireGuard	ChaCha20	Poly1305	Найкращий сучасний протокол для повсякденного використання
SoftEther	AES-256	Sha сімейсвто та HMAC	Повсякденне використання
PPTP	128-bit	Sha сімейство	Протокол тунелювання; використання обмежене, оскільки технологія застаріла
SSTP	AES-256	Sha сімейство	Протокол тунелювання Microsoft; для підключення Windows-пристроїв
L2TP/IPsec	AES-256	Sha сімейство	Немає причин використовувати, оскільки він поступається IKEv2 в усіх відношеннях

WireGuard вирізняється своєю простотою, високою швидкістю та ефективністю. Він впроваджує новаторський підхід до технології VPN, який робить його привабливим для користувачів, особливо на сучасних пристроях та в умовах обмежених мережевих ресурсів. Однак, вибір між OpenVPN та WireGuard також може залежати від конкретних потреб, рівня безпеки, інтеграції та використання у конкретній галузі.

2.2. Мережеві аномалії. Дослідження та аналіз.

Атакою на інформаційну систему називаються навмисні дії зловмисника, що використовують уразливості інформаційної системи і призводять до порушення доступності, цілісності та конфіденційності оброблюваної інформації [1]. Усунення вразливості інформаційної системи призводить до усунення і самої можливості реалізації атак. Існує три типи атак:

- Розвідка. Ці атаки включають ping sweeps, передачу DNS зони, розвідку за допомогою e-mail, сканування TCP або UDP портів і, можливо, аналіз загальнодоступних серверів з метою знаходження cgi-дір.
- Експлойт (exploit - використовувати у своїх інтересах, зловживати). Це комп'ютерна програма, фрагмент програмного коду або послідовність команд, що використовують уразливості в програмному забезпеченні та застосовуються для проведення атаки на обчислювальну систему [9]. Метою атаки може бути як захоплення контролю над системою (підвищення привілеїв), так і порушення її функціонування (DoS-атака). Порушники використовуватимуть переваги прихованих можливостей або помилок для отримання несанкціонованого доступу до системи.

- Відмова в обслуговуванні (Denial of Service, DoS). Під час такої атаки порушник намагається зруйнувати сервіс (або комп'ютер), перевантажити мережу, перевантажити центральний процесор або переповнити диск.

Моделі атак. Традиційна модель атаки будується за принципом “one to one” або “one to many”, тобто атака виходить з одного джерела. Розробники мережевих засобів захисту (міжмережевих екранів, систем виявлення атак тощо) орієнтовані саме на традиційну модель атаки. У різних точках захисту мережі встановлюються агенти (сенсори) системи захисту, які передають інформацію на центральну консоль управління. Це полегшує масштабування системи, забезпечує простоту віддаленого управління тощо. Однак така модель не справляється з розподіленими атаками.

У моделі розподіленої атаки використовуються інші принципи. На відміну від традиційної моделі в розподіленій моделі використовуються відносини “many to one” і “many to many” .

Розподілені атаки засновані на “класичних” атаках типу “відмова в обслуговуванні”, а точніше на їхній підмножині, відомій як Flood- або Storm-атаки (зазначені терміни можна перекласти як ‘”шторм”, “повені” або “лавина”). Сенс цих атак полягає в надсиланні великої кількості пакетів на атакований вузол. Атакований вузол може вийти з ладу, оскільки він "захлинеться” в лавині пакетів, що посилаються, і не зможе обробляти запити авторизованих користувачів. Однак у разі, якщо пропускна спроможність каналу до атакованого вузла перевищує пропускну спроможність атакуючого або атакований вузол некоректно налаштований, до “успіху” така атака не призведе. Але розподілена атака відбувається вже не з однієї точки Internet, а одразу з декількох, що призводить до різкого зростання трафіку і виведення атакованого вузла з ладу.

Набули поширення такі типи атак:

- Віддалене вторгнення (remote penetration). Атаки, які дають змогу реалізувати віддалене керування комп'ютером через мережу. Наприклад, NetBus або BackOrifice;
- Локальне вторгнення (local penetration). Атака, яка призводить до отримання несанкціонованого доступу до вузла, на якому вона запущена. Наприклад, GetAdmin;
- Віддалена відмова в обслуговуванні (remote denial of service). Атаки, які дають змогу порушити функціонування або перевантажити комп'ютер через Internet. Наприклад, Teardrop або trin00;

Локальна відмова в обслуговуванні (local denial of service). Атаки, які дають змогу порушити функціонування або перевантажити комп'ютер, на якому вони реалізуються. Прикладом такої атаки є “ворожий” аплет, який завантажує центральний процесор нескінченним циклом, що призводить до неможливості обробки запитів інших додатків.

Класифікації атак. Класифікацію атак можна розділити на кілька типів. Наприклад, класифікація на пасивні та активні, зовнішні та внутрішні, навмисні та ненавмисні.

Таблиця 2.4.

Характеристика мережових аномалійі

Тип аномалії	**Опис**	**Характеристика**
Альфа аномалія	Високий трафік типу “point-to-point”	Збільшення інтенсивності трафіку помітне байт/с, пакет/ через єдиний домінуючий шлях пересування джерело - призначення. Коротка аномалія (до 10 хвилин)
DoS,DDOS атака	Розподілена атака типу відмова в обслуговуванні атака на одну жертву	Викид у поданні трафіку на пакети/с, потоки/с, від безлічі джерел до однієї адреси призначення
Перевантаження	Незвичайно великий попит на конкретний	Стрибок у трафіку за потоками/с до

	мережовий ресурс або сервіс	однієї домінуючої IP-адреси, що домінує і домінуючого порту. Зазвичай короткочасна аномалія
Сканування мережі та портів	Сканування мережі за певними відкритими портами або сканування одного хоста за всіма портами з метою пошуку вразливостей	Стрибок у трафіку за потоками/с, з кількома пакетами в потоках від однієї домінуючої IP-адреси, що домінує
Мережевий черв`як	Шкідлива програма, яка самостійно поширюється мережею і використовує вразливості операційних систем	Викид у трафіку без домінуючої адреси призначення, але завжди з одним або кількома домінуючими портами призначення
Точка – мульти точка	Поширення контенту від одного сервера багатьом користувачам	Викид у пакетах, байтах від домінуючого джерела до кількох призначень, усі до одного (одних) добре відомих портів
Відключення	Мережеві неполадки, які спричиняють падіння в трафіку між однією парою джерело-призначення	Падіння в трафіку за пакетами, потоками і байтами зазвичай до нуля. Може бути довготривалим і включати всі потоки джерело-призначення від або до одного маршрутизатора
Зміна потоку	Незвичайне перемикання потоків трафіку з одного вхідного маршрутизатора на інший	Падіння в байтах або пакетах в одному потоці трафіку і викид в іншому. Може зачіпати кілька потоків трафіку

Найприроднішим і найпоширенішим чином можна класифікувати наявні актуальні аномалії за типом джерела або причини їх виникнення. Приклад такої класифікації наведено на (рис. 2.2.). У табл. 2.1 подано основні типи мережевих аномалій, їхній опис та основні характеристики. Наведена систематизація даних

про атаки та етапи їх реалізації дає необхідний базис для розуміння технологій виявлення атак.

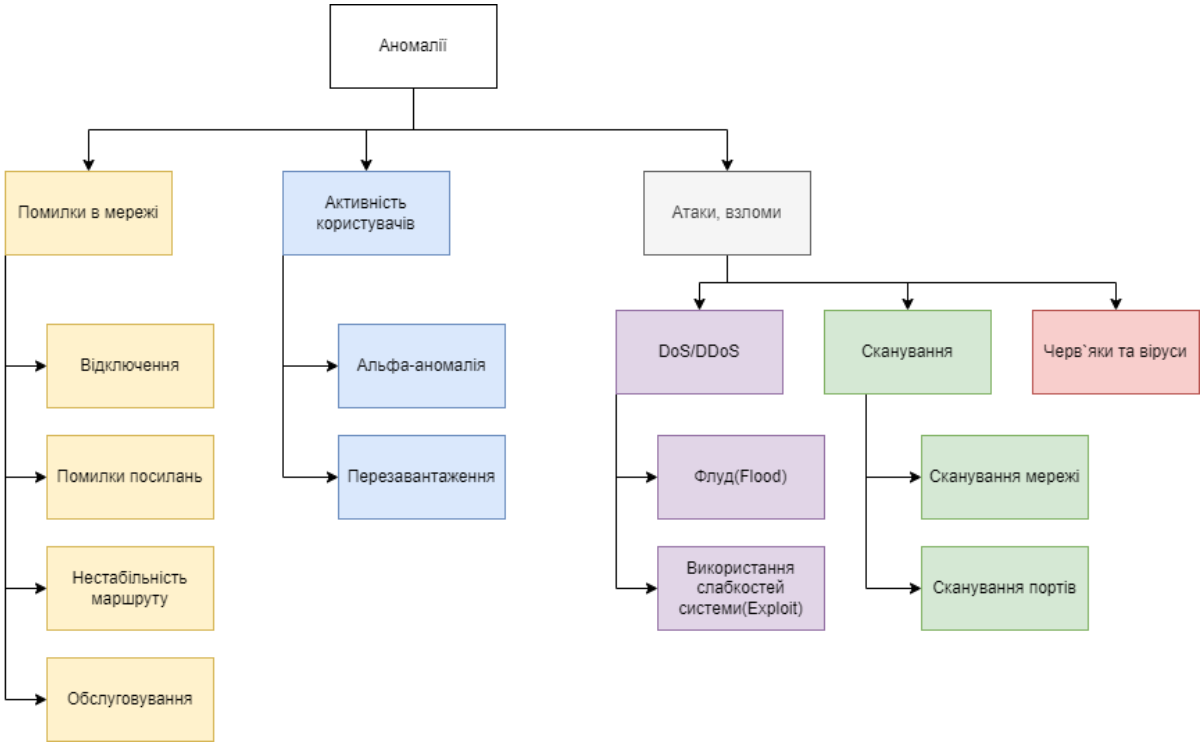


Рис. 2.2. Класифікація аномалій

Знаходження аномалій за допомогою нейроних мереж. Серед паралельних методів розпізнавання, що використовують евклідовий простір описів образів, можна виділити широкий клас методів, що заслуговує на окремий розгляд, - нейромережеві методи. В їхній основі лежать нейронні мережі – обчислювальні моделі, принцип функціонування яких схожий із мережами біологічних нейронів головного мозку. Завдяки запозиченню принципів організації біологічних структур мозку нейромережі демонструють багато їхніх властивостей, таких, як навчання на основі попереднього досвіду, вилучення істотних властивостей з інформації, що надходить, узагальнення наявних прецедентів на нові випадки. Можливості, що надаються нейронними мережами, були використані для розв'язання задач розпізнавання і класифікації образів у безлічі досліджень і прикладних розробок. У цьому розділі розглядаються деякі основні різновиди нейронних мереж, їхні можливості та способи навчання стосовно виявлення аномалій трафіку. Застосування нейронних мереж обумовлюється самою неформальною постановкою завдання - виявити аномальну поведінку. Ідея

полягає в тому, щоб, отримавши деяку тренерувальну множину параметрів вхід-вихід, що характеризує поведінку системи, дати мережі звикнути або навчитись до них. Виходом може бути деякий коефіцієнт нормальності або впевненості поведінки або один із параметрів системи. Якщо вихідні дані мають закономірності, то робиться припущення, що мережа здатна навчитися на них. Якщо у процесі роботи запропонований нейронною мережею вихід, за умови що він є деяким коефіцієнтом, потрапляє в небезпечну область або відрізняється від наявного в реальній системі, якщо це один із параметрів системи, то робиться висновок, що в системі є аномалія.

Для побудови шаблону поведінки користувача можуть використовуватися такі параметри, як час, коли він зазвичай працює, набір вузлів, з яких він починає робочу сесію, характеристики використання ресурсів системи тощо. Ці параметри оцифровуються і слугують входом у нейронну мережу зворотного поширення помилки (backpropagation neuralnetwork, BPNN), а виходом є коефіцієнт, що дорівнює нулю для користувача з нормальною поведінкою і рівний одиниці - з аномальною. Іншими словами, мережа тренується на парах типу (нормальні параметри, 0) і (аномальні параметри, 1). Оскільки для отримання аномальної поведінки треба було б би змусити користувача поводитися не так, як він звик, то аномальні дані генеруються випадково, що ускладнює інтерпретацію результатів щодо роботи на реальних даних.

В основі апарату нейронних мереж лежить принцип подібності штучного нейрона (ШН) біологічному прототипу. У найпростішому варіанті ІН - це бінарний пороговий елемент, який обчислює суму вхідних сигналів $z(t)$ і формує на виході сигнал 1, якщо ця сума перевищує певний поріг, і сигнал 0 в іншому випадку. Порогова функція є найбільш спрощеною активаційною функцією ІН. Найбільш точну роботу ІНС забезпечують функція гіперболічного тангенса, сигмоїдальна і логістична функції активації.

Сигнал надходить на вхідний шар рецепторних нейронів, кожен з яких пов'язаний з усіма елементами з яких пов'язаний з усіма елементами вихідного

шару з певними значеннями вагових коефіцієнтів w(i). Їхнє підсумовування призводить до збудження тих нейронів робочого шару, значення активаційної функції яких перевищило порогове значення.

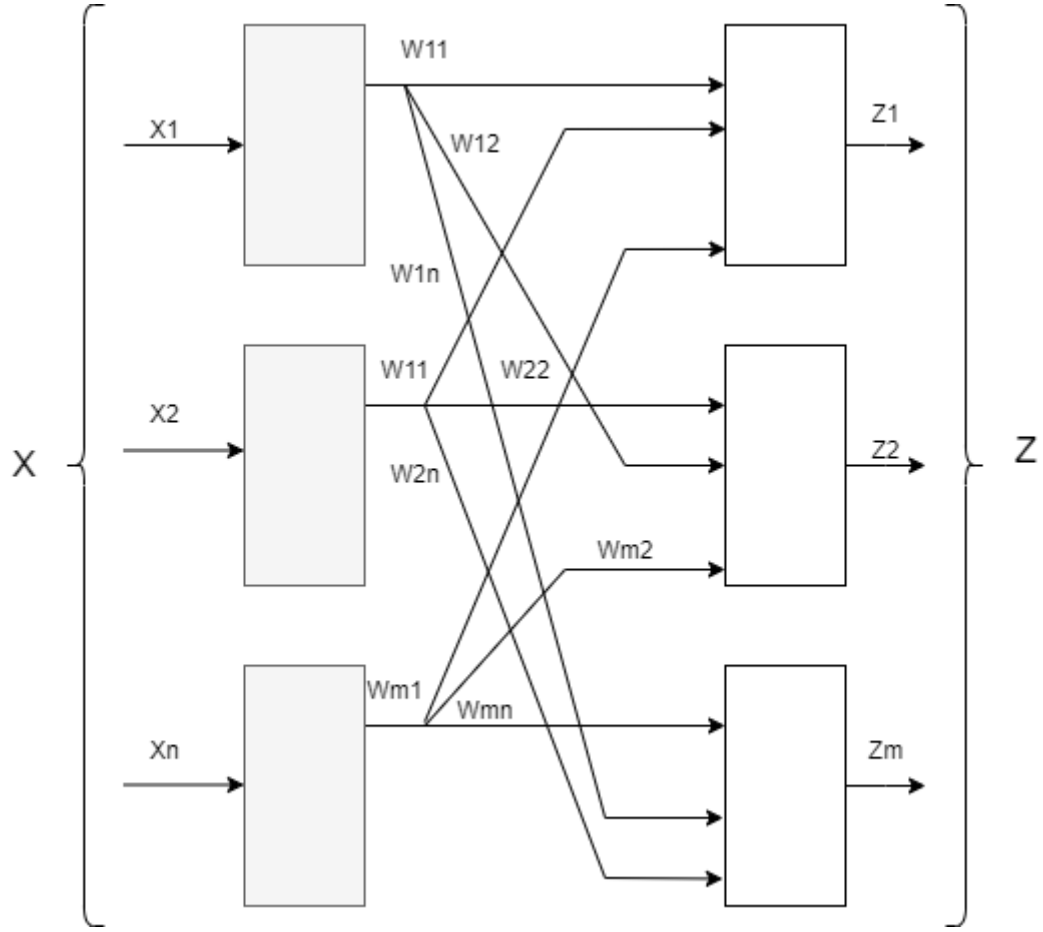


Рис. 2.3. Модель одношарової нейромережі.

Існує безліч різних моделей ІНС: з різною кількістю робочих (проміжних) шарів, зі зворотними зв'язками або без них, з повнозв'язними або довільно пов'язаними нейронами прихованих шарів. Головною перевагою нейронних мереж є можливість їх навчання з метою створення гнучких адаптивних систем управління. Зокрема апарат ІНС може успішно вирішувати завдання виявлення аномалій мережевої активності: виявлення вторгнень, порушення правил роботи в мережі та будь-яких нехарактерних для даної мережі дій, які неможливо виявити, використовуючи поширені системи виявлення вторгнень. Застосування адаптивної системи передбачає побудову профілю мережі та окремих користувачів у режимі нормального функціонування (відсутність атак та інших непередбачених дій). У цей період відбувається навчання ІНС, мережа

налаштовується на бажані параметри та запам'ятовує свій стан. Після введення системи управління в експлуатацію штучна нейронна мережа буде виявляти всі відхилення і порушення в роботі комп'ютерної мережі, що перевищують задане під час налаштування ШНМ порогове значення.

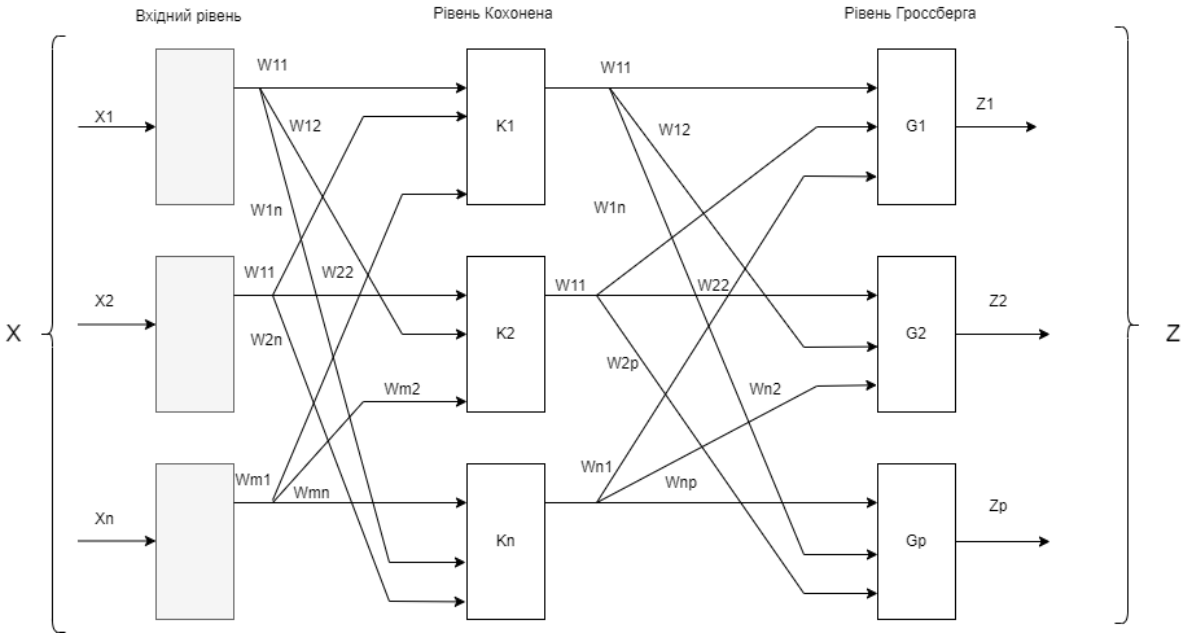


Рис. 2.4. Спрощена версія мережі зустрічного розповсюдження

Ключовою проблемою застосування нейромереж для вирішення завдань подібного типу є вибір параметрів і методу навчання ІНС. Оскільки некоректний вибір структури може призвести до неповного або неточного навчання мережі, потрапляння її в локальні мінімуми або до відсутності збіжності. Для створення систем виявлення вторгнень і виявлення мережевих аномалій одним із можливих варіантів є використання ІНС зустрічного поширення, що складається з двох шарів: шар ІН Кохонена і шар ІН Гроссберга. Перший шар має здатність витягувати з даних статистичні властивості, другий узагальнює й уточнює результати. Важливою особливістю є те, що подібні мережі використовують змішане навчання: шар Кохонена навчається без учителя, нейрони шару Гроссберга навчаються на результатах першого шару за допомогою комбінування зворотного поширення з навчанням Коші. Нормалізація векторів вхідних значень дає змогу отримати більш точні результати. Процес нормалізації являє собою

собою ділення кожної компоненти вхідного вектора на його довжину. Така операція перетворює вхідний вектор X на вектор одиничної
довжини Xn у m-вимірному просторі. У цій ІНС використовується механізм латерального гальмування (загострення вхідного сигналу). Якщо на шар нейронів, що містить латеральних зв'язків, подати вхідний вектор, що має невеликий максимум, то в процесі релаксації мережі здійснюється підвищення його контрастності (загострення). При великому значенні максимуму відбувається згладжування контрастності активаційною функцією. Така структура нейронної мережі дає змогу створювати ефективні засоби захисту комп'ютерних мереж з можливістю адаптації до мінливих умов функціонування самої мережі та вимог до її роботи.

Робота з нейронною мережею передбачає наявність таких етапів:

- збір і підготовка вихідних даних;
- побудова і навчання мережі;
- тестування мережі та аналіз результатів.

Використання перетворення Хафа для виявлення аномалій трафіку. У задачах комп'ютерного зору під час аналізу зображень найчастіше буває необхідно виділяти в зображенні різні криві (прямі, кола тощо). Для вирішення цього завдання існує ціле сімейство методів, заснованих на перетворенні Хафа. Теоретичні можливості цього перетворення дають змогу не обмежуватися площиною і дискретними кривими, його можна застосовувати для пошуку кривих у хмарі точок на площині або в багатовимірному просторі.

Перетворення Хафа - це метод, що дає змогу виділяти в двійковому зображенні графічні об'єкти теоретично будь-якої форми. Він ґрунтується на поданні шуканого об'єкта у вигляді параметричного рівняння, тому на практиці використовується для пошуку найпростіших форм - прямих, кіл, еліпсів тощо. Обчислювальна складність алгоритму перетворення швидко зростає зі збільшенням складності графічного об'єкта, тобто його аналітичного представлення.

Алгоритм методу. Вихідне зображення має являти собою собою безліч точок двох типів: фонових точок і точок інтересу. Розглянемо сімейство таких кривих, заданих параметричним рівнянням $F(a_1, a_2, \dots, a_n, x, y) = 0$, де x, y - координати на площині, a - параметри сімейства кривих. Ці параметри утворюють фазовий простір, кожна точка якого відповідає деякій кривій. У ньому вводиться дискретна сітка, що розбиває його на комірки, кожна з яких відповідає набору кривих із близькими значеннями параметрів. Фінальним кроком є обхід осередків простору Хафа та вибір максимальних значень, за які "проголосувало" найбільше точок зображення, що і дає параметри для рівнянь шуканого об'єкта. Як приклад розглянемо параметричне рівняння прямої в полярній системі координат:

$$F(R, \theta, x, y) = x\cos\ \theta + y\sin\ \theta - R,$$

де R - довжина перпендикуляра, опущеного на пряму з початку координат; θ - кут між перпендикуляром до прямої та віссю x. Точка (x_0, y_0) на площині зображення відповідає безлічі ліній, кожну з яких можна виразити через різні ρ та θ. Кожна з цих ліній відповідає точкам на площині (ρ, θ) які, будучи зібраними, разом утворюють криву форми. Аналізуючи вихідне зображення і відповідний йому фазовий простір йому фазовий простір і вибравши комірку з максимальним значенням лічильника кривих, що потрапили, можна виділити відповідну їй пряму на вихідному зображенні.

Використання методу виявлення мережевих аномалій. У задачі виявлення аномалій трафіку IP-мереж перетворення Хафа має прикладне значення. В основі одного з методів виявлення аномалій лежить вейвлет-перетворення трафіку, за якого він розкладається на масштабні рівні, кожен з яких містить свій діапазон частот. Аналізуючи кожен із цих рівнів статистичними методами, знаходяться точки зміни таких параметрів, як дисперсія, середнє значення та характеристики самоподібності. Якщо такі точки зміни знайдено на кількох масштабних рівнях і приблизно однаково локалізовані у часі, то можна зробити висновок про наявність аномалії в даній ділянці. Для автоматизованого прийняття рішення про

наявність аномалії в цьому випадку й використовують перетворення Хафа. Для цього знайдені точки зміни подаються у вигляді точок інтересу на бінарному зображенні, і завдання зводиться до виявлення вертикальних прямих за певною кількістю знайдених на різних рівнях точок, приблизно однаково локалізованих у часі.

2.3. Висновки до розділу.

У цьому розділі ми провели глибокий аналіз двох ключових аспектів інформаційної безпеки: захисту віддалених підключень та виявлення мережових аномалій. Загальною висновок з цього розділу є необхідність інтеграції сучасних засобів захисту та аналізу для створення комплексного підходу до забезпечення інформаційної безпеки в сучасних мережевих середовищах. Отримані знання визначать подальші кроки у розробці та впровадженні заходів безпеки для підтримання стабільності та надійності корпоративних мереж.

РОЗДІЛ 3. РОЗРОБКА ЗАСОБУ ЗАХИСТУ ВІДДАЛЕНИХ ПІДЛКЛЮЧЕНЬ.

3.1. Вибір технологій розробки

Вибір правильних технологій є вирішальним етапом у створенні ефективного та надійного засобу захисту. У цьому розділі ми ретельно розглянемо різні технології розробки, їхні переваги та недоліки в контексті створення засобу для захисту віддалених підключень.

Розглянемо питання вибору мов програмування, фреймворків, інструментів для тестування та інших ключових аспектів, які визначатимуть характеристики, ефективність та безпеку розробленого засобу. Також, зосередимо увагу на засобах контролю якості коду та техніках безпеки програмного забезпечення, що впливатимуть на загальну стійкість та надійність створеного рішення.

В даному розділі буде реалізований засіб моніторингу трафіку клієнта для аналізу небезпек зі сторони корпоративного працівника. Засіб буде розроблено, як графічний додаток для операційної системи Linux. Для швидкої, розширюваної та легко читаємої реалізації використав мову програмування Python. Для імплементації графічної оболонки користувача використав найпопулярнішу бібліотеку Python Tkinter. Розглянемо більш детально технології.

Python - це мова комп'ютерного програмування, яку часто використовують для створення веб-сайтів і програмного забезпечення, автоматизації завдань та аналізу даних. Python - це мова загального призначення, тобто її можна використовувати для створення різноманітних програм і вона не спеціалізована для вирішення якихось конкретних завдань. Ця універсальність, а також

дружність до початківців, зробила її однією з найпоширеніших мов програмування сьогодні. Найпопулярніша по версії PYPL мова програмування(рис. 3.1.), опитування розробників Stack Overflow за 2022 рік показало, що Python є четвертою за популярністю мовою програмування, а респонденти стверджують, що вони використовують Python майже 50 відсотків часу у своїй роботі. Результати опитування також показали, що Python зрівнявся за популярністю з Rust: 18% розробників, які ще не використовують цю мову, вже заявили, що зацікавлені у вивченні Python.

Існує два різних типи комп'ютерних мов: скомпільовані та інтерпретовані. Python - це інтерпретована мова. Коли ви запускаєте Python, ви запускаєте інтерпретатор, який складається з внутрішнього компілятора та віртуальної машини Python (PVM). Python спочатку компілюється, що приховано від програміста, а потім інтерпретується.

Наприклад, щоб запустити файл Python з назвою hello.py, ви можете ввести $python hello.py. Ця команда запускає інтерпретатор Python, який надсилає файл hello.py внутрішньому компілятору для перекладу у двійковий код.

Це не машинна мова, яку може зрозуміти і виконати процесор вашого комп'ютера, а проміжна мова, яку розуміє PVM. Саме PVM виконує код.

Worldwide, Dec 2023 :

Rank	Change	Language	Share	1-year trend
1		Python	28.09 %	+0.1 %
2		Java	15.81 %	-0.9 %
3		JavaScript	8.93 %	-0.5 %
4	↑	C/C++	6.8 %	+0.1 %
5	↓	C#	6.64 %	-0.3 %
6		PHP	4.6 %	-0.6 %
7		R	4.53 %	+0.5 %
8		TypeScript	2.81 %	+0.0 %
9	↑	Swift	2.8 %	+0.7 %
10	↓	Objective-C	2.33 %	+0.2 %

Рис. 3.1. Рейтинг популярності мови програмування PYPL

Python став широко використовуваним і улюбленим завдяки цілому ряду позитивних якостей. Ось деякі з переваг Python.

- Простота використання: Python має простий, стислий і зрозумілий синтаксис. Програма на Python виглядає дуже схожою на звичайну англійську мову і добре читається. Це робить програми на Python легкими для читання та налагодження. Структури управління Python інтуїтивно зрозумілі та прості у використанні. Крім того, Python має динамічну типізацію, тому немає необхідності оголошувати тип кожної змінної. З цих причин Python є однією з найбільш ефективних і продуктивних мов.
- М'яка крива навчання: Python є однією з найпростіших мов для вивчення і є хорошим варіантом для людей, які вчаться програмувати. Програмісти, які переходять на Python з таких мов, як C або Java, можуть швидко досягти максимальної ефективності. Пакет Python містить корисне інтегроване середовище розробки та навчання (IDLE).
- Універсальність: Python - це гнучка мова загального призначення, яка повністю підтримує як процедурне, так і об'єктно-орієнтоване програмування. Завдяки вбудованим та стороннім пакетам вона підходить для широкого спектру завдань. Вона домінує у сферах науки про дані та машинного навчання. Вона також широко використовується для внутрішньої веб-розробки та Інтернету речей (IoT). Навіть коли вона не є найкращим вибором для конкретного завдання, зазвичай вона все одно залишається життєздатним варіантом. Крім того, код Python можна вбудовувати в проекти, написані іншими мовами, такими як C++, і код з інших мов можна вбудовувати в Python.
- Ефективна для швидкої розробки: Оскільки Python простий у використанні і не потребує компіляції, розробка програм займає

менше часу. Програми на Python зазвичай набагато коротші, ніж еквівалентні програми на інших мовах. Це чудовий вибір для швидкого створення прототипів у середовищі швидкої розробки програмного забезпечення.

- Справжня портативність: Величезною перевагою Python є те, що її можна написати один раз і запускати будь-де. Python не потрібно компілювати заздалегідь, тому користувачі запускають справжню програму на Python, а не виконуваний файл Python. Програма не компілюється, поки не буде запущена на виконання, використовуючи специфічний для платформи PVM. Це означає, що будь-яка програма на Python потенційно може працювати на будь-якій системі, яка підтримує Python.
- Відсутність процесу компіляції: Python є інтерпретованою мовою, і програми автоматично компілюються під час виконання. Програму можна запускати одразу після написання. Немає окремого компілятора, немає трудомісткого етапу компіляції і немає непрозорих помилок компілятора. Програми на Python легко писати, налагоджувати та поступово змінювати.
- Автоматичний розподіл пам'яті: У Python немає вказівників і розробникам не потрібно виділяти вільне місце в пам'яті. Python виділяє пам'ять автоматично, а збирач сміття переробляє пам'ять відкинутих об'єктів. Це означає, що розробникам не потрібно турбуватися про скрайблери, витоки пам'яті, невірні посилання на вказівники або розмір кожного об'єкта.
- Широкі вбудовані об'єкти та бібліотеки: Python має велику кількість вбудованих складних об'єктів, включаючи списки, множини та словники, подібні до записів. Кожен з цих об'єктів має набір методів, що дозволяє легко їх обробляти. Крім того, Python має велику бібліотеку, що містить десятки тисяч функцій. Ці пакети можна

використовувати для мережевих комунікацій, веб-інтеграції, обробки даних та взаємодії з апаратним забезпеченням. Це значно прискорює написання програм, оскільки багато необхідних процедур вже написані.

- Доступність сторонніх бібліотек: На додаток до великої вбудованої бібліотеки Python, розробники мають доступ до багатьох безкоштовних зовнішніх бібліотек. Ці сторонні бібліотеки легко імпортувати та встановлювати за допомогою менеджера пакетів Python pip. Пакунки можна завантажити зі сховища Python Package Index (PyPI). PyPI також дозволяє розробникам публікувати власні пакунки.
- Відкритий вихідний код та вільне використання: Всі випуски Python доступні безкоштовно під ліцензією з відкритим вихідним кодом. Python можна навіть змінювати та розповсюджувати безкоштовно. Це значно знижує витрати на розробку. Щоб дізнатися більше про ліцензування Python, відвідайте сайт документації Python.
- Велика база користувачів: Python має велику, активну та захоплену спільноту користувачів. Тут легко знайти навчальні матеріали та інші ресурси, поставити запитання, знайти роботу, найняти додаткових розробників і познайомитися з іншими Python-програмістами.

Незважаючи на свої численні переваги, Python також має кілька помітних недоліків. Ось деякі з недоліків Python.

- Не дуже швидка: Python працює набагато повільніше, ніж більш ефективні мови, такі як C та Java. Python інтерпретується та динамічно типізується, тому компілятор часу виконання має багато роботи. Він повинен постійно перевіряти тип кожної змінної. Це означає, що Python - не найкращий вибір для сценаріїв, де швидкість є критично важливою.

- Вимагає багато пам'яті: Python не оптимізовано для зменшення пам'яті. Вона може використовувати в десять разів більше оперативної пам'яті, ніж програма, написана більш економною мовою. Однак це частково є компромісом в обмін на гнучкість і простоту використання. Крім того, збирач сміття Python не може зібрати всі відкинуті ресурси одразу, що зменшує обсяг доступної пам'яті. Python не є хорошим вибором для середовищ з обмеженою пам'яттю.
- Важко уникнути помилок під час виконання: Python не компілюється до часу виконання і динамічно типізується. Тому багато проблем, які в іншому випадку були б виявлені компілятором, не з'являються, поки програма не буде запущена. Це може бути щось просте, як синтаксична помилка, але може включати і такі проблеми, як спроба додати ціле число і рядок разом.
- Не дуже популярний у мобільних та десктопних програмах: Через те, що Python є дещо повільною мовою і використовує багато пам'яті, вона не набула значного поширення в мобільному просторі. Існує кілька інструментів розробки для мобільних додатків, але вони більш обмежені, ніж фреймворки для інших мов. Ситуація трохи краща в клієнтських десктопах, але для фронтенд-додатків Python все ще не надто популярний.
- Не оптимізований для доступу до баз даних: Працювати з базами даних у Python складніше, ніж у деяких інших додатках. Python не має потужного, якісного, простого у використанні інтерфейсу, як, наприклад, Java Database Connectivity (JDBC). Його все ще можна використовувати, якщо читання і запис до бази даних є відносно простими. Але це не найкращий вибір для додатків, які мають складну взаємодію з великою корпоративною базою даних.
- Відсутність підтримки багатопоточності: Через свою архітектуру Python не підтримує багатопоточність. Замість цього він використовує багатопроцесорну обробку, де кожна "нитка" виконується в окремому

процесі Python. Це покладається на нагляд операційної системи за плануванням та балансуванням процесів, і може не давати однаково хороших результатів.

- Схильність до надмірного або неправильного використання: Простота Python є однією з його сильних сторін, але в деяких ситуаціях вона може стати несподіваною слабкістю. Через те, що вона настільки проста у використанні, її часто зловживають для завдань, де вона не є однією з найкращих альтернатив. Python чудово підходить для швидкої розробки та створення прототипів, але це може спокусити організації нехтувати належними принципами розробки програмного забезпечення.

Другий інструмент, який був використаний у ході розробки – це масивна графічна бібліотека Tkinter (рис. 3.2.) яка є частиною Python та нічого додатково завантажувати не потрібно. Дуже простий синтаксис.

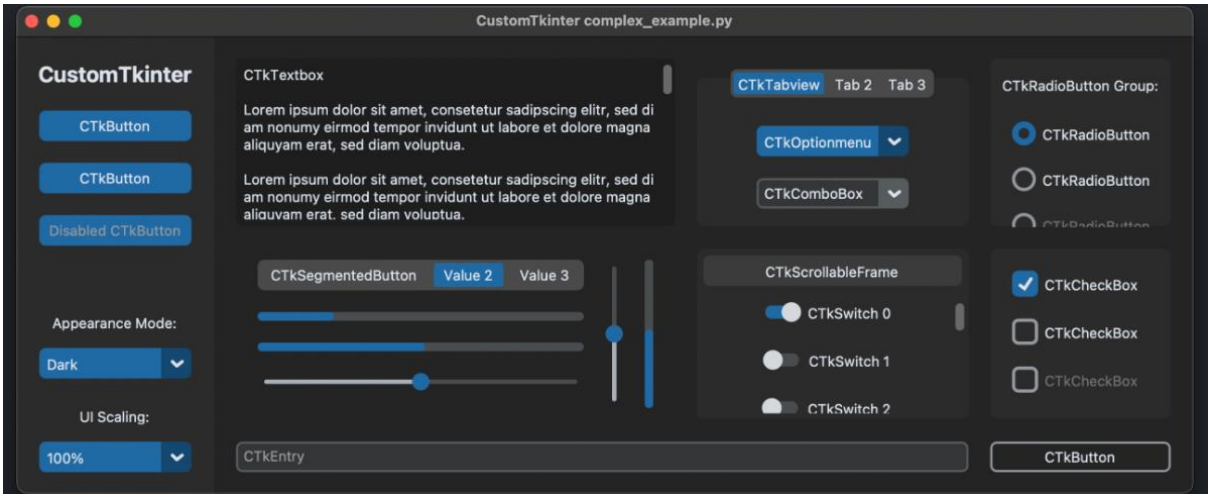


Рис. 3.2. Вікно з Tkinter віджетами

Текстовий віджет надзвичайно потужний і з ним дуже легко працювати. Віджет полотна також дуже простий і потужний. З мого досвіду, жоден інший інструментарій не забезпечує таке ж поєднання простоти і потужності, як ці два віджети. Використовує власні віджети на mac та windows. Tk надійний, з невеликою кількістю крос-платформних особливостей. Я вважаю, що три менеджери геометрії Tkinter - pack, place і grid - набагато потужніші і простіші у використанні ніж у інших графічних бібліотеках. Його переваги:

- Стабільний.
- Широко портований, можливо портувати віджети в інші бібліотеки. Також є можливість інтегрувати бібліотеки UI для малювання сучасного інтерфейсу
- Простий API, який легко вивчити. Низький поріг входу дає користувачу цікавий та швидкий процес вивчення даної бібліотеки.

Мінуси:

- Існує певне занепокоєння щодо швидкості роботи Tkinter. Більшість викликів Tkinter форматується як команда Tcl (рядок) і інтерпретується Tcl, з якої виконуються виклики Tk. Це теоретичне уповільнення, спричинене послідовним виконанням двох інтерпретованих мов, рідко спостерігається на практиці, і більшість реальних

За статистикою, є найбільш популярним фреймворком для розробки графічних додатків серед інших конкурентів (рис 3.3):

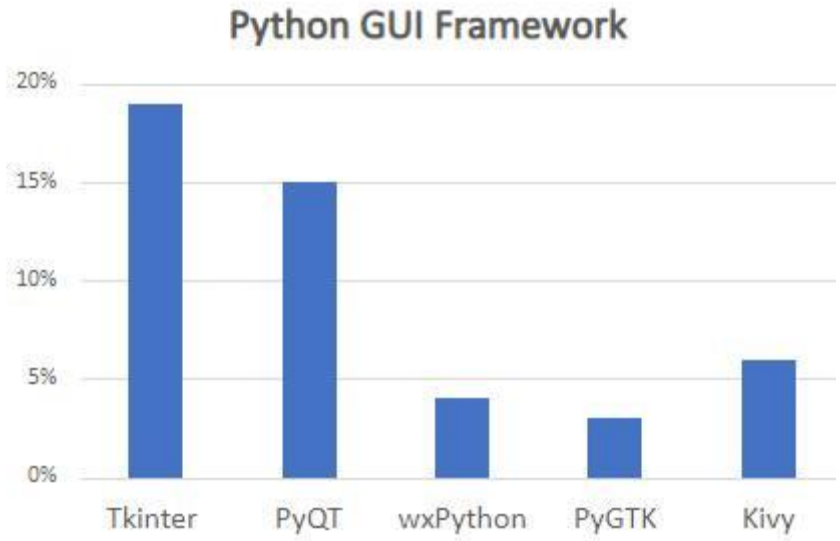


Рис. 3.3. Рейтинг використання GUI бібліотек.

Для аналізу трафіку у проєкті було використано Scapy фреймворк. Scapy - це потужна інтерактивна програма та бібліотека для маніпулювання пакетами на основі Python.

Вона здатна підробляти або декодувати пакети великої кількості протоколів, надсилати їх по дроту, перехоплювати, зберігати або читати їх за

допомогою pcap-файлів, зіставляти запити і відповіді та багато іншого. Він призначений для швидкого створення прототипів пакетів, використовуючи значення за замовчуванням, які працюють. Він може легко впоратися з більшістю класичних завдань, таких як сканування, трасування, зондування, модульні тести, атаки або виявлення мережі (він може замінити hping, 85% nmap, arpspoof, arp-sk, arping, tcpdump, wireshark, p0f і т.д.). Він також дуже добре справляється з багатьма іншими специфічними завданнями, з якими більшість інших інструментів не можуть впоратися, наприклад, відправка недійсних кадрів, ін'єкція власних кадрів 802.11, комбінування технік (VLAN hopping+ARP cache poisoning, декодування VoIP на захищеному WEP каналі, ...) тощо. Також він є кросс-платформенним тому наш додаток буде функціонувати на Windows/Linux/Mac. Вигляд роботи фреймворку з IP пакетом (рис 3.4):

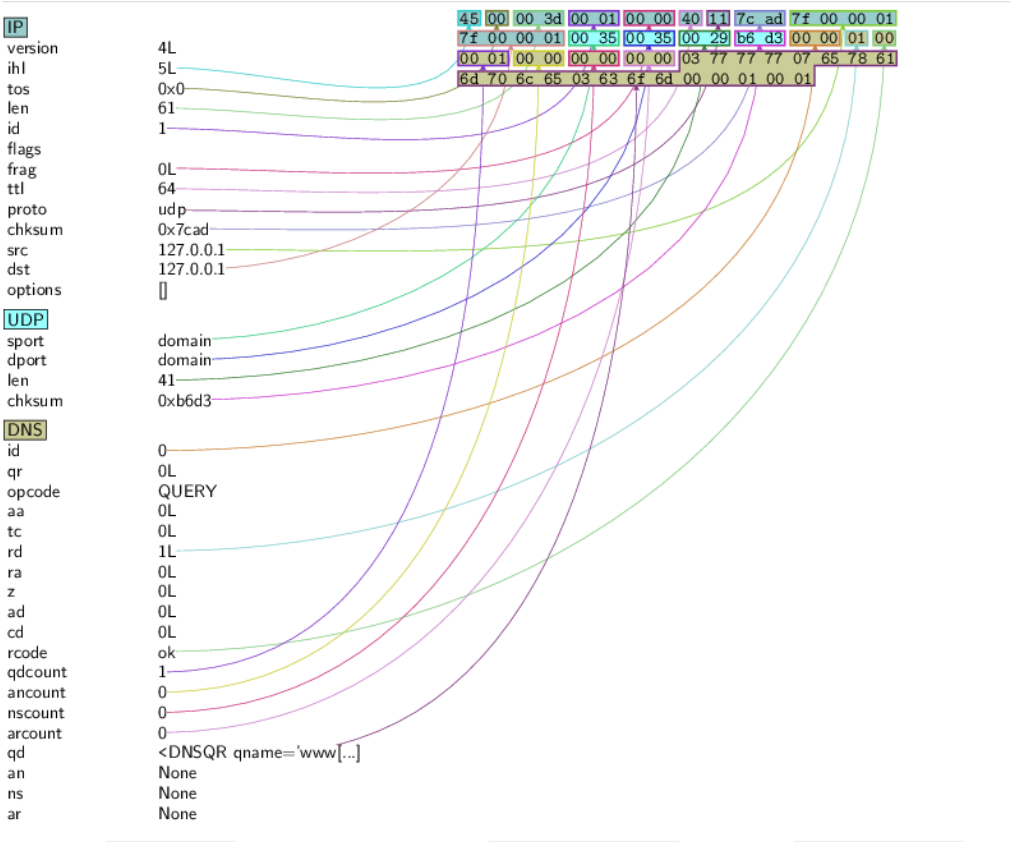


Рис. 3.4. Візуалізація IP пакету.

Також однією з цікавих функцій фреймворку є трасування шляху, ми можемо побачити на графіку який шлях проходить наш пакет від точки А до

точки Б. Використовуючи функцію world_trace() ми намалюємо карту з шляхом пакету(рис. 3.5.).

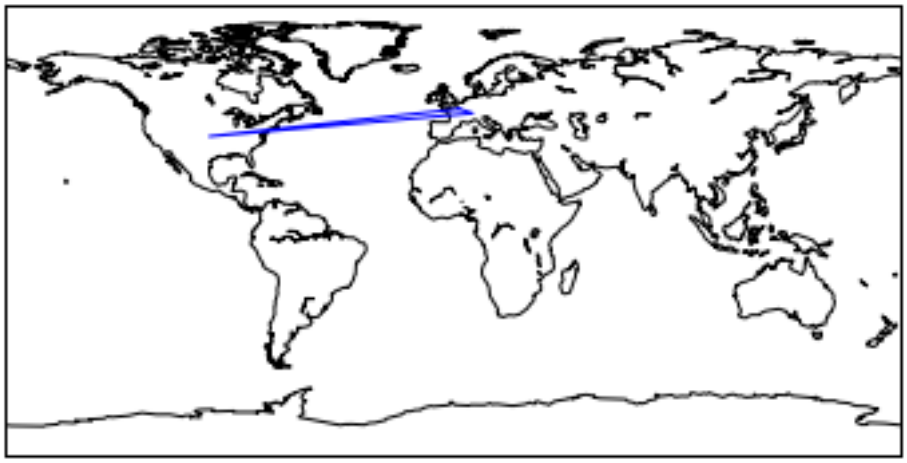


Рис. 3.5. Результат трасування пакету

В додатку моніторингу та аналізу трафіку, буде використаний стандартний формат IP пакету (рис. 3.6.):

Example : Default Values for IP

```
>>> ls(IP)
version    : BitField             = (4)
ihl        : BitField             = (None)
tos        : XByteField           = (0)
len        : ShortField           = (None)
id         : ShortField           = (1)
flags      : FlagsField           = (0)
frag       : BitField             = (0)
ttl        : ByteField            = (64)
proto      : ByteEnumField        = (0)
chksum     : XShortField          = (None)
src        : Emph                 = (None)
dst        : Emph                 = ('127.0.0.1')
options    : IPoptionsField       = ('')
```

Рис. 3.6. Структура IP пакету.

Для реалізації звітності, яка буде доставлятись на сервер компанії від користувача. Будемо використовувати бібліотеку socket, та звичайний сервер, який приймає дані через інтерфейс сокетів. Всі дані будуть зашифровані за

допомогою алгоритму AES-256, тривіальний конвеєр шифрування та дешифрування з 32 байтним ключем (рис. 3.7.)

```
KEY = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits) for _ in range(32))
ENCRYPTOR = AES.new(KEY.encode("utf8"), AES.MODE_CBC, 'This is an IV-12'.encode("utf8"))
DECRYPTOR = AES.new(KEY.encode("utf8"), AES.MODE_CBC, 'This is an IV-12'. encode("utf8"))

def aes_encrypt(plaintext):
    ciphertext = ENCRYPTOR.encrypt(plaintext)
    return ciphertext

def aes_decrypt(ciphertext):
    plaintext = DECRYPTOR.decrypt(ciphertext)
    return plaintext

def encrypt(msg):
    encrypted = aes_encrypt(msg)
    return encrypted

def decrypt(encrypted_msg):
    decrypted = aes_decrypt(encrypted_msg)
    return decrypted
```

Рис. 3.7. Прототим системи шифрування та дешифрування.

3.2. Демонстрація функціоналу програмного додатку

Програмний додаток має назву PA(Packets analyzer) та має 3 основних функції (рис. 3.8.):

- Аналіз трафіку та виявлення аномалій.
- Створення графіку найбільш відвідуваним веб ресурсам
- Створення точки розположення веб серверу ресурсу на карті у представлені: карта вулиць чи супутник.
- Демонстрація логів користувачу з метою знаходження програмних помилок, так як програмний модуль є відкритим до розробки іншими користувачами.

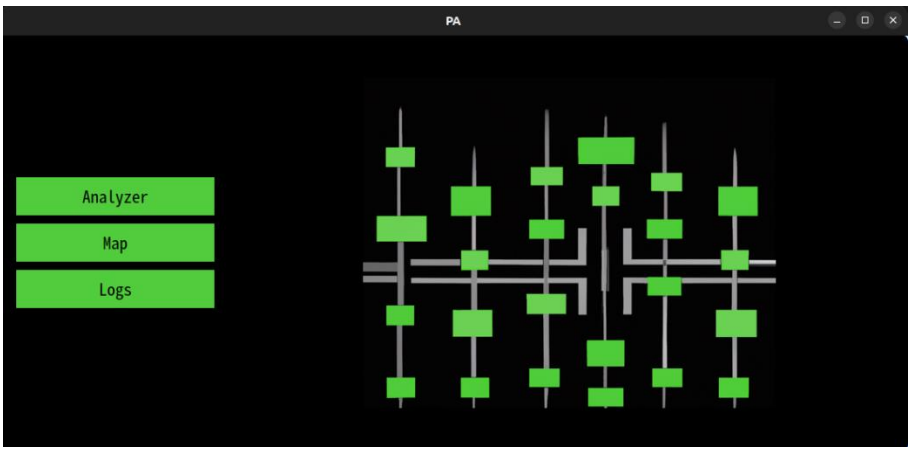


Рис. 3.8. Меню аналізатора

Функція Analyzer, представлена на (рис. 3.9), включає в себе перехоплення усього мережевого трафіку користувача, використовуючи tun/tap інтерфейси. Під час перехоплення IP пакетів, відправлених та отриманих користувачем, аналізатор розбирає кожен пакет та виводить основну інформацію з його хедера на екран. Також, частину зібраного трафіку він записує у вигляді дампу пакетів, який, в свою чергу, піддається аналізу кожні 15 хвилин. Згодом створюється набір даних для подальшого аналізу цього дампу на сервері, використовуючи нейромережу, яка виявляє потенційні аномалії в мережевому трафіку.

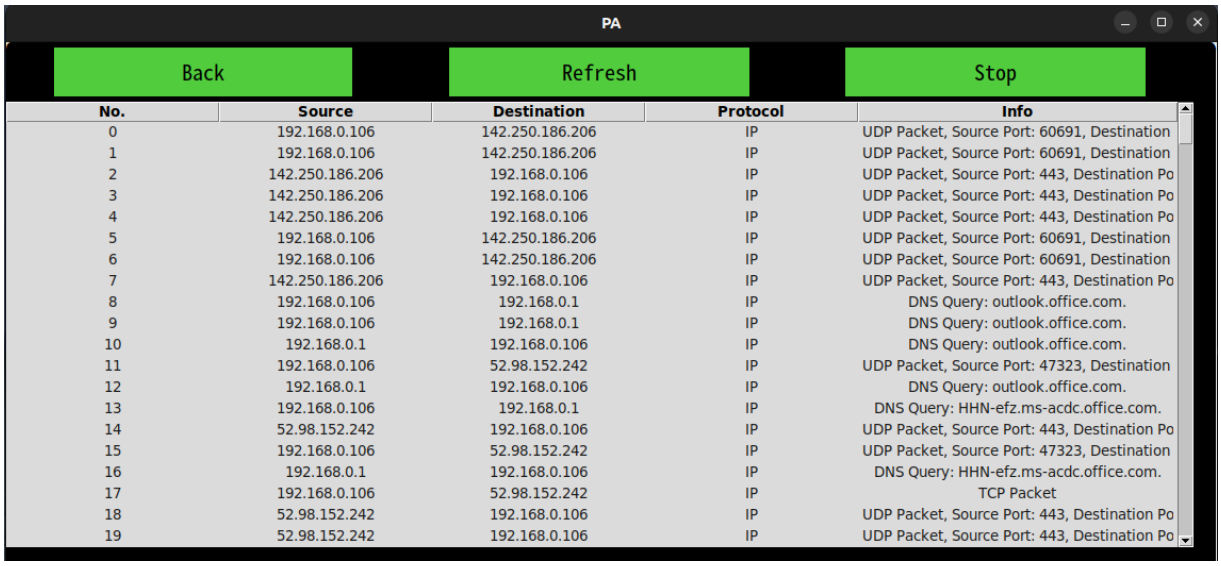
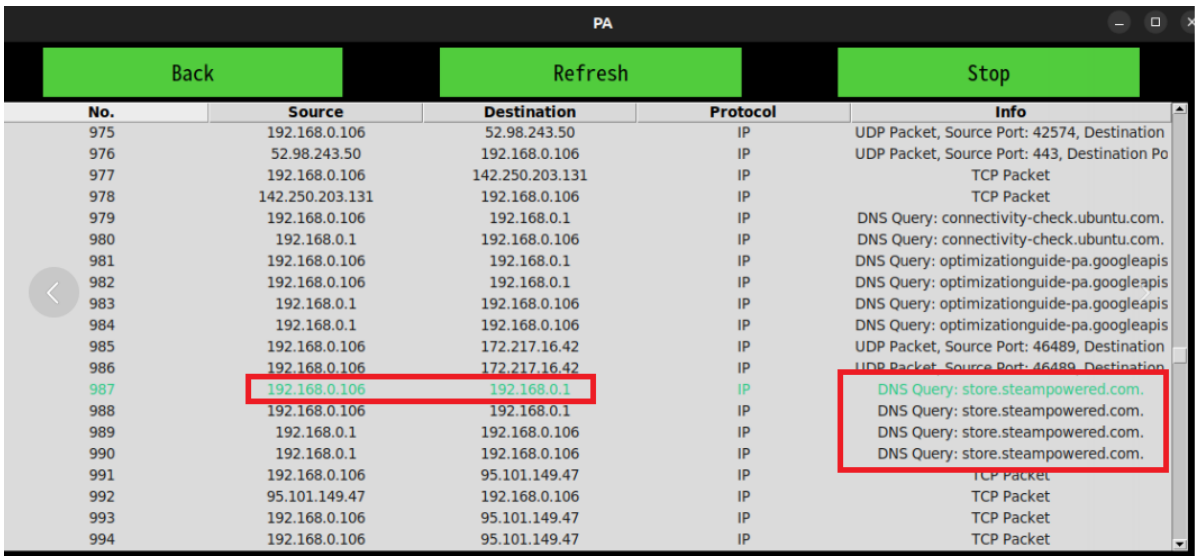


Рис. 3.9. Аналізатор

Важливою частиною перехоплених пакетів є те, що ми можемо створювати моніторинговий набір даних та спостерігати на який веб ресурс заходив користувач в певний проміжок часу (рис. 3.10.), аналізуючи DNS запити.



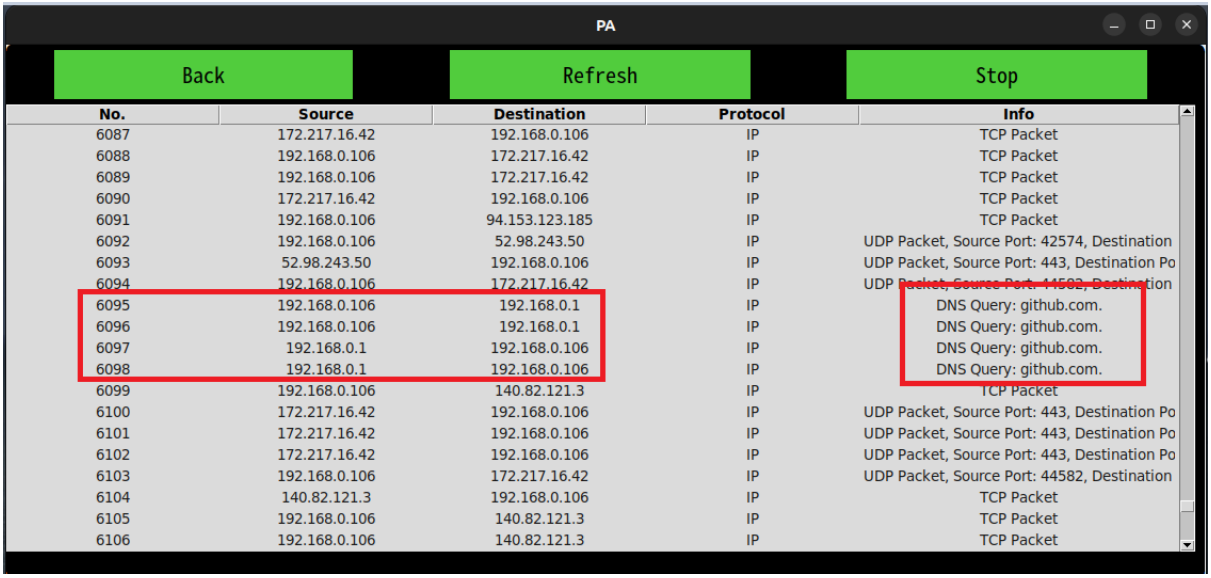


Рис. 3.10. – 3.11. DNS відповідь клієнту.

У аналізатора доступні 3 основні функції керування вікном:

- Повернутись назад до меню
- Очистити панель трафіку
- Зупинити/Запустити панель трафіку

При натискані кнопки Refresh панель трафіку очистить і все почнеться спочатку (рис. 3.12.)

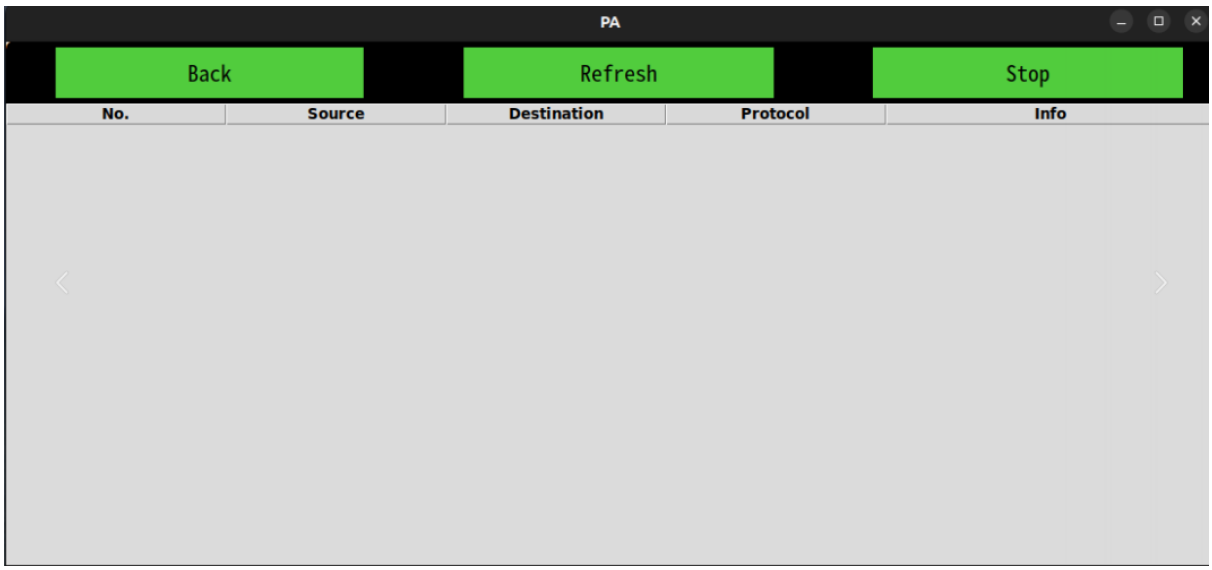


Рис 3.12 Оновлення панелі

Наступна ключова функція програмного додатку – це функціонал map. При натисканні кнопки "Map" у меню, користувачу відкриється графічне відображення веб-ресурсів, які аналізатор запам'ятовує з трафіку користувача. Графік інформує про найбільш популярні ресурси, що були відвідані користувачем (рис. 3.13.).

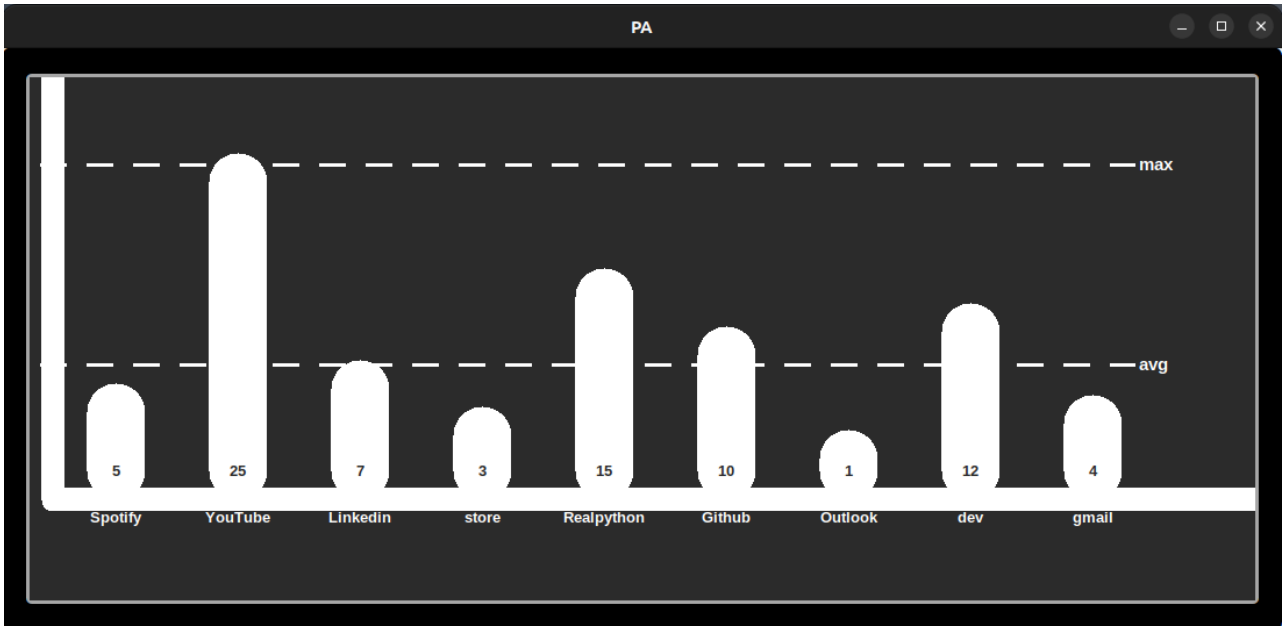


Рис. 3.13. Графік відвідувань ресурсів.

Остання доступна користувачеві функція аналізу – це відображення місцезнаходження веб-сервера за його IP-адресою. Наприклад, користувач може

перевірити місце розташування свого веб-сервера.(рис. 3.14).

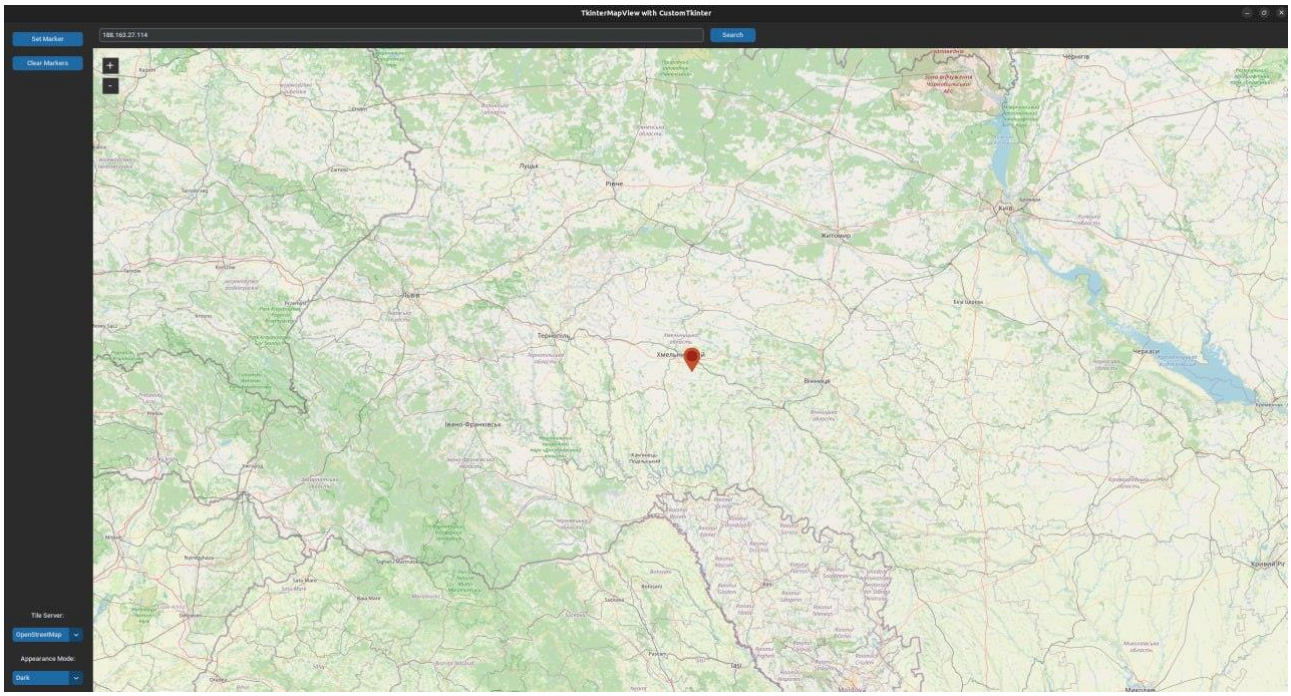


Рис. 3.14. Мапа з сервером веб ресурсу.

3.3. Висновки до розділу

У третьому розділі кваліфікаційної роботи були отримані такі результати:

Був реалізований програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі:

- Вибір технологій реалізації продукту. Було обрано мову програмування Python, за допомогою якої розроблялось низькорівневе ПЗ для моніторингу трафіку користувача в корпоративній мережі.
- Детальний опис структури програмного засобу у вигляді графічного додатку для ОС Linux.
- Розгорнутий опис усіх етапів реалізації засобу захисту віддалених підключень.

– Тестування готового програмного продукту, використовуючи мережу клієнта та пакети, які аналізатор перехоплює для аналізу.

Програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі пройшов успішне тестування, підтверджуючи його ефективність та придатність для практичного використання. В рамках верифікації були проведені тести з використанням чотирьох різних віддалених клієнтів корпоративної інфраструктури.

Результати тестування вказують на те, що засіб успішно виявляє потенційні загрози для безпеки мережі користувача.

Висновок вказує на те, що розроблений програмний засіб має перевагу над аналогічними системами за трьома ключовими аспектами: його реалізація є простою та ефективною, він є відкритим для розробки іншими користувачами, а також враховує особливості корпоративного середовища.

РОЗДІЛ 4. ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Забруднення атмосфери є однією з найбільш нагальних екологічних проблем сучасного світу. Цей термін відноситься до викиду шкідливих речовин у повітря, які негативно впливають на навколишнє середовище та здоров'я людей. Головними джерелами забруднення атмосфери є промисловість, автотранспорт, а також спалювання відходів.

Промислові викиди, включаючи випуск вуглекислого газу, сірчаних та азотних оксидів, є значним чинником забруднення атмосфери. Ці гази впливають не тільки на якість повітря, але й сприяють ефекту парникового газу, що призводить до глобального потепління.

Транспортні засоби, особливо автомобілі, є ще одним важливим джерелом забруднення. Вихлопні гази містять низку токсичних речовин, включаючи вуглеводні, оксиди азоту та вуглекислий газ. Ці забруднювачі сприяють утворенню смогу та загрожують здоров'ю людей, викликаючи респіраторні захворювання та інші проблеми зі здоров'ям.

Спалювання відходів, яке часто використовується як метод утилізації твердих відходів, також є джерелом забруднення атмосфери. Під час спалювання виділяються токсичні гази та частинки, які забруднюють повітря та можуть надавати шкідливий вплив на довкілля та здоров'я людей.

Отже, забруднення атмосфери є складною проблемою, яка вимагає уваги та дій на рівнях як місцевих, так і глобальних. Вирішення цієї проблеми є ключовим кроком для забезпечення здорового майбутнього для нашої планети та всіх її мешканців.

Види забруднювачів. Атмосфера Землі постійно піддається впливу різноманітних забруднювачів, що мають значний вплив на якість повітря та здоров'я людей. До основних видів забруднювачів належать:

- Вуглекислий газ (CO2): Це головний парниковий газ, викид якого в атмосферу в основному здійснюється через спалювання викопного палива (вугілля, нафта, газ) для енергетики, транспорту та інших промислових процесів. Вуглекислий газ сприяє глобальному потеплінню та кліматичним змінам.
- Сірчані оксиди (SOx): Основним джерелом цих газів є спалювання вугілля та нафтопродуктів. Вони сприяють утворенню кислотних дощів, які можуть негативно впливати на рослинність, водні екосистеми та будівельні матеріали.
- Азотні оксиди (NOx): Ці гази в основному викидаються автомобілями та електростанціями. Вони є одними з основних компонентів фотохімічного смогу та також сприяють утворенню кислотних дощів.
- Чадний газ (CO): Цей токсичний газ утворюється внаслідок неповного згоряння вуглеводнів. Чадний газ може викликати серйозні проблеми зі здоров'ям, включаючи отруєння та навіть смерть при високій концентрації.
- Свинець: Цей токсичний метал часто потрапляє в атмосферу через промислові процеси та раніше широко використовувався як добавка в автомобільному пальному. Свинець може викликати серйозні проблеми зі здоров'ям, особливо у дітей, включаючи розвиткові та нервові розлади.
- Інші токсичні речовини: До цієї категорії належать різноманітні хімічні речовини, такі як бензопірен, аміак, хлоровані вуглеводні та інші, які можуть впливати на здоров'я людини та навколишнє середовище.

Вплив цих забруднювачів на здоров'я людини може бути дуже серйозним, включаючи респіраторні захворювання, серцево-судинні проблеми, алергії та навіть рак. Окрім того, вони негативно впливають на екосистеми, зменшуючи біорізноманіття та порушуючи природні процеси. Охорона атмосфери від цих забруднювачів є ключовою для забезпечення сталого розвитку та здоров'я населення.

1. Забруднення атмосфери має глибокі та далекосяжні наслідки, які впливають на навколишнє середовище, здоров'я людини, а також на глобальні кліматичні зміни.
2. Глобальне потепління: Викиди вуглекислого газу та інших парникових газів сприяють ефекту парникового газу, що веде до підвищення середньої температури Землі. Це викликає зміни клімату, підвищення рівня моря, екстремальні погодні умови, такі як сильніші шторми та спекотні хвилі.
3. Кислотні дощі: Викиди сірчаних та азотних оксидів утворюють кислоти у вологому середовищі атмосфери, що призводить до випадіння кислотних дощів. Це може завдати шкоди лісам, озерам та річкам, порушуючи екологічний баланс та вбиваючи морське життя.
4. Проблеми зі здоров'ям: Забруднення повітря пов'язане з широким спектром проблем зі здоров'ям. Воно може викликати або погіршувати захворювання дихальної системи, такі як астма та хронічна обструктивна хвороба легень, а також сприяти розвитку серцево-судинних захворювань. Довготривале вплив токсичних речовин може підвищувати ризик раку.
5. Вплив на екосистеми: Забруднення атмосфери має шкідливий вплив на флору та фауну. Воно може вплинути на здоров'я та репродуктивну здатність диких тварин, порушити біологічні процеси рослин та призвести до зменшення біорізноманіття.

6. Зміни у рослинному покриві: Забруднення може впливати на зростання рослин, знижуючи їх продуктивність та стійкість до хвороб і шкідників. Це може вплинути на сільське господарство, зменшуючи врожайність та якість продукції.

7. Нарушення фотосинтезу: Деякі забруднювачі можуть блокувати сонячне світло, необхідне для фотосинтезу, тим самим знижуючи здатність рослин до виробництва кисню та органічних сполук.

Ці наслідки підкреслюють необхідність ефективних заходів щодо контролю забруднення атмосфери та зменшення викидів забруднювачів для захисту навколишнього середовища та здоров'я людей.

Забруднення атмосфери є однією з найбільш серйозних екологічних проблем, з якою стикається сучасне суспільство. Його наслідки — глобальне потепління, кислотні дощі, проблеми зі здоров'ям, зміни в екосистемах — підкреслюють критичну важливість захисту якості повітря. Щоб зменшити забруднення атмосфери, необхідні спільні зусилля урядів, промисловості, наукових спільнот та кожного окремого громадянина.

Покращення якості повітря вимагає впровадження чистіших технологій, зменшення викидів від промислових об'єктів та транспортних засобів, а також збільшення використання відновлюваних джерел енергії. Не менш важливою є роль освіти та залучення громадськості до вирішення цієї проблеми, виховання свідомості про екологічно відповідальну поведінку та споживання.

Кожен з нас може внести свій вклад у захист навколишнього середовища: від зміни щоденних звичок, таких як використання громадського транспорту або велосипеда замість автомобіля, до участі в місцевих екологічних ініціативах. Також важливим є підтримка політик, які спрямовані на зменшення забруднення та захист довкілля.

Усвідомлення та активність кожної людини відіграє ключову роль у забезпеченні чистого та безпечного майбутнього для нас самих та наступних поколінь. Від нас залежить, чи буде наша планета здоровою та гостинною для життя, тому важливо діяти сьогодні, аби зберегти землю для майбутнього.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблений програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі, котрий, аналізуючи кожний пакет, який відправить та прийме користувач, забезпечує створення безпечного корпоративного середовища та аналітичних даних активності користувача, включаючи мережеві аномалії користувача. У процесі виконання роботи отримані такі результати:

1. Досліджено сучасну архітектуру корпоративної мережі. Також в рамках задачі було проаналізовано кожний рівень мережевої моделі для виявлення можливих загроз, що дало змогу обрати алгоритми запобігання небезпекам на 2 та 3 рівні моделі TCP/IP.
2. Були досліджено і проаналізовано існуючі методи захисту віддалених підключень до корпоративної мережі. В результаті аналізу знайшов універсальний алгоритм шифрування трафіку між працівником та корпоративною мережою.

Було розроблено програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі. Програмний засіб моніторингу та захисту мережевого трафіку у корпоративній мережі пройшов успішне тестування, підтверджуючи його ефективність та придатність для практичного використання. В рамках верифікації були проведені тести з використанням чотирьох різних віддалених клієнтів корпоративної інфраструктури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Splunk vs ArcSight: a Head-to-Head Comparison. [Електронний ресурс]. – Режим доступу: https://kinneygroup.com/blog/splunk-vs-arcsight/

2. What is Splunk? Tools, Architecture, Advantages [Електронний ресурс]. – Режим доступу: https://www.knowledgehut.com/blog/database/what-is-splunk

3. Security Analysis of ChaCha20-Poly1305 AEAD. KDDI Research, Inc. February 2017. [Електронний ресурс]. – Режим доступу: https://www.cryptrec.go.jp/exreport/cryptrec-ex-2601-2016.pdf

4. AES Encryption Algorithms. [Електронний ресурс]. – Режим доступу: https://xilinx.github.io/Vitis_Libraries/security/2019.2/guide_L1/internals/aes.html

5. A Guide To Anomaly Detection and its Role in the Network. [Електронний ресурс]. – Режим доступу: https://www.pingplotter.com/wisdom/article/anomaly-detection-role-in-networks/

6. What Is User Datagram Protocol (UDP). [Електронний ресурс]. – Режим доступу: https://www.fortinet.com/resources/cyberglossary/user-datagram-protocol-udp

7. The Complete Guide to FTP, FTPS, SFTP, and SCP. [Електронний ресурс]. – Режим доступу: https://www.integrate.io/blog/the-complete-guide-to-ftp-ftps-sftp-and-scp/#FTP

8. What is Transmission Control Protocol TCP/IP. [Електронний ресурс]. – Режим доступу: https://www.fortinet.com/resources/cyberglossary/tcp-ip

9. What Is A Firewall? . [Електронний ресурс]. – Режим доступу: https://www.fortinet.com/resources/cyberglossary/firewall

10. Unveiling the World of Servers: What They Are and How They Work. [Електронний ресурс]. – Режим доступу: https://codeinstitute.net/global/blog/what-is-a-server/

11. Exploring the Enterprise Network Infrastructure. [Електронний ресурс]. – Режим доступу: https://www.pearsonhighered.com/assets/samplechapter/1/5/8/7/1587132117.pdf

12. Enterprise Networking Explained: Types, Concepts & Trends. [Електронний ресурс]. – Режим доступу: https://www.bmc.com/blogs/enterprise-networking/

13. Parallels RAS Integrates Well with Complex Network Infrastructures. [Електронний ресурс]. – Режим доступу: https://www.parallels.com/blogs/ras/enterprise-network/

14. Top 6 enterprise network trends. [Електронний ресурс]. – Режим доступу: planet.com/management/what-is-an-enterprise-network/

15. CustomTkinter is a python UI-library based on Tkinter. [Електронний ресурс]. – Режим доступу: https://github.com/TomSchimansky/CustomTkinter, https://customtkinter.tomschimansky.com/

16. Python Tkinter Table. [Електронний ресурс]. – Режим доступу: https://pythonguides.com/python-tkinter-table-tutorial/

17. Creating Network Interfaces and Checking Interface Types. . [Електронний ресурс]. – Режим доступу: https://www.baeldung.com/linux/create-check-network-interfaces

18. A simple VPN (tunnel with tun device) demo and some basic concepts. [Електронний ресурс]. – Режим доступу: https://lxd.me/a-simple-vpn-tunnel-with-tun-device-demo-and-some-basic-concepts

19. What is enterprise network security? [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/network-layer/enterprise-networking/

20. CRIME: Setting the stage for session hijacking attacks. [Електронний ресурс]. – Режим доступу: https://venafi.com/blog/what-session-hijacking/

21. How to prevent session hijacking attacks. [Електронний ресурс]. – Режим доступу: https://venafi.com/blog/what-session-hijacking/

22. SSH tunneling, or 'port forwarding'. [Електронний ресурс]. – Режим доступу: https://www.cloudflare.com/learning/access-management/what-is-ssh/

23. Socket Programming in Python. [Електронний ресурс]. – Режим доступу: https://realpython.com/python-sockets/#echo-server

24. Scapy. [Електронний ресурс]. – Режим доступу: https://scapy.readthedocs.io/en/latest/usage.html#interactive-tutorial

25. Python for Security and Networking. Third Edition. José Manuel Ortega. [Текст] : Network Scripting and Packet Sniffing with Python.

26. How Huff Model works. [Електронний ресурс]. – Режим доступу: https://pro.arcgis.com/en/pro-app/latest/tool-reference/business-analyst/understanding-huff-model.htm

27. Network Anomaly. . [Електронний ресурс]. – Режим доступу: https://www.ibm.com/docs/en/qradar-common?topic=extensions-network-anomaly

28. Machine Learning Approaches to Network Anomaly Detection. [Електронний ресурс]. – Режим доступу: https://www.usenix.org/legacy/event/sysml07/tech/full_papers/ahmed/ahmed_html/sysml07CR_07.html

29. TCP Characteristics. [Електронний ресурс]. – Режим доступу: https://www.guru99.com/tcp-ip-model.html

30. Таненбаум.Е - Комп'ютерні мережі (Класика Computer Science)-2012. [Текст]: 8.6.1. Ipsec - стр 862.

Додаток А

Лістинг головного класу сніферу, який перехоплює трафік

```
class AsyncSniffer(object):
    """
    Sniff packets and return a list of packets.

    Args:
        count: number of packets to capture. 0 means infinity.
        store: whether to store sniffed packets or discard them
        prn: function to apply to each packet. If something is returned, it
             is displayed.
             --Ex: prn = lambda x: x.summary()
        session: a session = a flow decoder used to handle stream of packets.
                 --Ex: session=TCPSession
                 See below for more details.
        filter: BPF filter to apply.
        lfilter: Python function applied to each packet to determine if
                 further action may be done.
                 --Ex: lfilter = lambda x: x.haslayer(Padding)
        offline: PCAP file (or list of PCAP files) to read packets from,
                 instead of sniffing them
        quiet:   when set to True, the process stderr is discarded
                 (default: False).
        timeout: stop sniffing after a given time (default: None).
        L2socket: use the provided L2socket (default: use conf.L2listen).
        opened_socket: provide an object (or a list of objects) ready to use
                       .recv() on.
        stop_filter: Python function applied to each packet to determine if
                     we have to stop the capture after this packet.
                     --Ex: stop_filter = lambda x: x.haslayer(TCP)
        iface: interface or list of interfaces (default: None for sniffing
               on all interfaces).
        monitor: use monitor mode. May not be available on all OS
        started_callback: called as soon as the sniffer starts sniffing
                          (default: None).

    The iface, offline and opened_socket parameters can be either an
    element, a list of elements, or a dict object mapping an element to a
    label (see examples below).

    For more information about the session argument, see
    https://scapy.rtfd.io/en/latest/usage.html#advanced-sniffing-sniffing-sessions

    Examples: synchronous
      >>> sniff(filter="arp")
      >>> sniff(filter="tcp",
      ...       session=IPSession,  # defragment on-the-flow
      ...       prn=lambda x: x.summary())
```

Продовження додатку А

```
    >>> sniff(lfilter=lambda pkt: ARP in pkt)
    >>> sniff(iface="eth0", prn=Packet.summary)
    >>> sniff(iface=["eth0", "mon0"],
    ...       prn=lambda pkt: "%s: %s" % (pkt.sniffed_on,
    ...                                   pkt.summary()))
    >>> sniff(iface={"eth0": "Ethernet", "mon0": "Wifi"},
    ...       prn=lambda pkt: "%s: %s" % (pkt.sniffed_on,
    ...                                   pkt.summary()))

  Examples: asynchronous
    >>> t = AsyncSniffer(iface="enp0s3")
    >>> t.start()
    >>> time.sleep(1)
    >>> print("nice weather today")
    >>> t.stop()
  """

  def __init__(self, *args, **kwargs):
      # type: (*Any, **Any) -> None
      # Store keyword arguments
      self.args = args
      self.kwargs = kwargs
      self.running = False
      self.thread = None  # type: Optional[Thread]
      self.results = None  # type: Optional[PacketList]

  def _setup_thread(self):
      # type: () -> None
      # Prepare sniffing thread
      self.thread = Thread(
          target=self._run,
          args=self.args,
          kwargs=self.kwargs,
          name="AsyncSniffer"
      )
      self.thread.daemon = True

  def _run(self,
           count=0,  # type: int
           store=True,  # type: bool
           offline=None,  # type: Any
           quiet=False,  # type: bool
           prn=None,  # type: Optional[Callable[[Packet], Any]]
           lfilter=None,  # type: Optional[Callable[[Packet], bool]]
           L2socket=None,  # type: Optional[Type[SuperSocket]]
           timeout=None,  # type: Optional[int]
           opened_socket=None,  # type: Optional[SuperSocket]
           stop_filter=None,  # type: Optional[Callable[[Packet], bool]]
```

Продовження додатку А

```
iface=None,  # type: Optional[_GlobInterfaceType]
             started_callback=None,  # type: Optional[Callable[[], Any]]
             session=None,  # type: Optional[_GlobSessionType]
             **karg  # type: Any
             ):
        # type: (...) -> None
        self.running = True
        self.count = 0
        lst = []
        # Start main thread
        # instantiate session
        if not isinstance(session, DefaultSession):
            session = session or DefaultSession
            session = session()
        # sniff_sockets follows: {socket: label}
        sniff_sockets = {}  # type: Dict[SuperSocket, _GlobInterfaceType]
        if opened_socket is not None:
            if isinstance(opened_socket, list):
                sniff_sockets.update(
                    (s, "socket%d" % i)
                    for i, s in enumerate(opened_socket)
                )
            elif isinstance(opened_socket, dict):
                sniff_sockets.update(
                    (s, label)
                    for s, label in opened_socket.items()
                )
            else:
                sniff_sockets[opened_socket] = "socket0"
        if offline is not None:
            flt = karg.get('filter')

            if isinstance(offline, str):
                # Single file
                offline = [offline]
            if isinstance(offline, list) and \
                    all(isinstance(elt, str) for elt in offline):
                # List of files
                sniff_sockets.update((PcapReader(  # type: ignore
                    fname if flt is None else
                    tcpdump(fname,
                            args=["-w", "-"],
                            flt=flt,
                            getfd=True,
                            quiet=quiet)
                ), fname) for fname in offline)
            elif isinstance(offline, dict):
                # Dict of files
```

Продовження додатку А

```
 sniff_sockets.update((PcapReader(  # type: ignore
                    fname if flt is None else
                    tcpdump(fname,
                            args=["-w", "-"],
                            flt=flt,
                            getfd=True,
                            quiet=quiet)
                ), label) for fname, label in offline.items())
            elif isinstance(offline, (Packet, PacketList, list)):
                # Iterables (list of packets, PacketList..)
                offline = IterSocket(offline)
                sniff_sockets[offline if flt is None else PcapReader(
                    tcpdump(offline,
                            args=["-w", "-"],
                            flt=flt,
                            getfd=True,
                            quiet=quiet)
                )] = offline
            else:
                # Other (file descriptors...)
                sniff_sockets[PcapReader(  # type: ignore
                    offline if flt is None else
                    tcpdump(offline,
                            args=["-w", "-"],
                            flt=flt,
                            getfd=True,
                            quiet=quiet)
                )] = offline
        if not sniff_sockets or iface is not None:
            # The _RL2 function resolves the L2socket of an iface
            _RL2 = lambda i: L2socket or resolve_iface(i).l2listen()  # type:
Callable[[_GlobInterfaceType], Callable[..., SuperSocket]]  # noqa: E501
            if isinstance(iface, list):
                sniff_sockets.update(
                    (_RL2(ifname)(type=ETH_P_ALL, iface=ifname, **karg),
                     ifname)
                    for ifname in iface
                )
            elif isinstance(iface, dict):
                sniff_sockets.update(
                    (_RL2(ifname)(type=ETH_P_ALL, iface=ifname, **karg),
                     iflabel)
                    for ifname, iflabel in iface.items()
                )
            else:
                iface = iface or conf.iface
                sniff_sockets[_RL2(iface)(type=ETH_P_ALL, iface=iface,
                                          **karg)] = iface
```

Продовження додатку А

```
        # Get select information from the sockets
        _main_socket = next(iter(sniff_sockets))
        select_func = _main_socket.select
        nonblocking_socket = getattr(_main_socket, "nonblocking_socket", False)
        # We check that all sockets use the same select(), or raise a warning
        if not all(select_func == sock.select for sock in sniff_sockets):
            warning("Warning: inconsistent socket types ! "
                    "The used select function "
                    "will be the one of the first socket")

        close_pipe = None  # type: Optional[ObjectPipe[None]]
        if not nonblocking_socket:
            # select is blocking: Add special control socket
            from scapy.automaton import ObjectPipe
            close_pipe = ObjectPipe[None]()
            sniff_sockets[close_pipe] = "control_socket"  # type: ignore

            def stop_cb():
                # type: () -> None
                if self.running and close_pipe:
                    close_pipe.send(None)
                self.continue_sniff = False
            self.stop_cb = stop_cb
        else:
            # select is non blocking
            def stop_cb():
                # type: () -> None
                self.continue_sniff = False
            self.stop_cb = stop_cb

        try:
            if started_callback:
                started_callback()
            self.continue_sniff = True

            # Start timeout
            if timeout is not None:
                stoptime = time.monotonic() + timeout
            remain = None

            while sniff_sockets and self.continue_sniff:
                if timeout is not None:
                    remain = stoptime - time.monotonic()
                    if remain <= 0:
                        break
                sockets = select_func(list(sniff_sockets.keys()), remain)
                dead_sockets = []
```

Продовження додатку А

```
for s in sockets:
                    if s is close_pipe:  # type: ignore
                        break
                    # The session object is passed the socket to call recv() on,
                    # and may perform additional processing (ip defrag, etc.)
                    try:
                        packets = session.recv(s)
                        # A session can return multiple objects
                        for p in packets:
                            if lfilter and not lfilter(p):
                                continue
                            p.sniffed_on = sniff_sockets[s]
                            # post-processing
                            self.count += 1
                            if store:
                                lst.append(p)
                            if prn:
                                result = prn(p)
                                if result is not None:
                                    print(result)
                            # check
                            if (stop_filter and stop_filter(p)) or \
                                    (0 < count <= self.count):
                                self.continue_sniff = False
                                break
                    except EOFError:
                        # End of stream
                        try:
                            s.close()
                        except Exception:
                            pass
                        dead_sockets.append(s)
                        continue
                    except Exception as ex:
                        msg = " It was closed."
                        try:
                            # Make sure it's closed
                            s.close()
                        except Exception as ex2:
                            msg = " close() failed with '%s'" % ex2
                        warning(
                            "Socket %s failed with '%s'." % (s, ex) + msg
                        )
                        dead_sockets.append(s)
                        if conf.debug_dissector >= 2:
                            raise
                        continue
                # Removed dead sockets
```

Продовження додатку А

```
for s in dead_sockets:
                    del sniff_sockets[s]
                    if len(sniff_sockets) == 1 and \
                            close_pipe in sniff_sockets:  # type: ignore
                        # Only the close_pipe left
                        del sniff_sockets[close_pipe]  # type: ignore
        except KeyboardInterrupt:
            pass
        self.running = False
        if opened_socket is None:
            for s in sniff_sockets:
                s.close()
        elif close_pipe:
            close_pipe.close()
        self.results = PacketList(lst, "Sniffed")

    def start(self):
        # type: () -> None
        """Starts AsyncSniffer in async mode"""
        self._setup_thread()
        if self.thread:
            self.thread.start()

    def stop(self, join=True):
        # type: (bool) -> Optional[PacketList]
        """Stops AsyncSniffer if not in async mode"""
        if self.running:
            try:
                self.stop_cb()
            except AttributeError:
                raise Scapy_Exception(
                    "Unsupported (offline or unsupported socket)"
                )
            if join:
                self.join()
                return self.results
            return None
        else:
            raise Scapy_Exception("Not running ! (check .running attr)")

    def join(self, *args, **kwargs):
        # type: (*Any, **Any) -> None
        if self.thread:
            self.thread.join(*args, **kwargs)


@conf.commands.register
def sniff(*args, **kwargs):
```

Продовження додатку А

```
 # type: (*Any, **Any) -> PacketList
    sniffer = AsyncSniffer()
    sniffer._run(*args, **kwargs)
    return cast(PacketList, sniffer.results)


sniff.__doc__ = AsyncSniffer.__doc__


@conf.commands.register
def bridge_and_sniff(if1,  # type: _GlobInterfaceType
                     if2,  # type: _GlobInterfaceType
                     xfrm12=None,  # type: Optional[Callable[[Packet],
Union[Packet, bool]]]  # noqa: E501
                     xfrm21=None,  # type: Optional[Callable[[Packet],
Union[Packet, bool]]]  # noqa: E501
                     prn=None,  # type: Optional[Callable[[Packet], Any]]
                     L2socket=None,  # type: Optional[Type[SuperSocket]]
                     *args,  # type: Any
                     **kargs  # type: Any
                     ):
    # type: (...) -> PacketList
    """Forward traffic between interfaces if1 and if2, sniff and return
    the exchanged packets.

    :param if1: the interfaces to use (interface names or opened sockets).
    :param if2:
    :param xfrm12: a function to call when forwarding a packet from if1 to
        if2. If it returns True, the packet is forwarded as it. If it
        returns False or None, the packet is discarded. If it returns a
        packet, this packet is forwarded instead of the original packet
        one.
    :param xfrm21: same as xfrm12 for packets forwarded from if2 to if1.

    The other arguments are the same than for the function sniff(),
    except for offline, opened_socket and iface that are ignored.
    See help(sniff) for more.
    """
    for arg in ['opened_socket', 'offline', 'iface']:
        if arg in kargs:
            log_runtime.warning("Argument %s cannot be used in "
                                "bridge_and_sniff() -- ignoring it.", arg)
            del kargs[arg]

    def _init_socket(iface,  # type: _GlobInterfaceType
                     count,  # type: int
                     L2socket=L2socket  # type: Optional[Type[SuperSocket]]
                     ):
```

Продовження додатку А

```
# type: (...) -> Tuple[SuperSocket, _GlobInterfaceType]
        if isinstance(iface, SuperSocket):
            return iface, "iface%d" % count
        else:
            if not L2socket:
                iface = resolve_iface(iface or conf.iface)
                L2socket = iface.l2socket()
            return L2socket(iface=iface), iface
    sckt1, if1 = _init_socket(if1, 1)
    sckt2, if2 = _init_socket(if2, 2)
    peers = {if1: sckt2, if2: sckt1}
    xfrms = {}
    if xfrm12 is not None:
        xfrms[if1] = xfrm12
    if xfrm21 is not None:
        xfrms[if2] = xfrm21

    def prn_send(pkt):
        # type: (Packet) -> None
        try:
            sendsock = peers[pkt.sniffed_on or ""]
        except KeyError:
            return
        if pkt.sniffed_on in xfrms:
            try:
                _newpkt = xfrms[pkt.sniffed_on](pkt)
            except Exception:
                log_runtime.warning(
                    'Exception in transformation function for packet [%s] '
                    'received on %s -- dropping',
                    pkt.summary(), pkt.sniffed_on, exc_info=True
                )
                return
            else:
                if isinstance(_newpkt, bool):
                    if not _newpkt:
                        return
                    newpkt = pkt
                else:
                    newpkt = _newpkt
        else:
            newpkt = pkt
        try:
            sendsock.send(newpkt)
        except Exception:
            log_runtime.warning('Cannot forward packet [%s] received on %s',
                                pkt.summary(), pkt.sniffed_on, exc_info=True)
    if prn is None:
```

Продовження додатку А

```
   prn = prn_send
    else:
        prn_orig = prn

        def prn(pkt):
            # type: (Packet) -> Any
            prn_send(pkt)
            return prn_orig(pkt)

    return sniff(opened_socket={sckt1: if1, sckt2: if2}, prn=prn,
                 *args, **kargs)


@conf.commands.register
def tshark(*args, **kargs):
    # type: (Any, Any) -> None
    """Sniff packets and print them calling pkt.summary().
    This tries to replicate what text-wireshark (tshark) would look like"""

    if 'iface' in kargs:
        iface = kargs.get('iface')
    elif 'opened_socket' in kargs:
        iface = cast(SuperSocket, kargs.get('opened_socket')).iface
    else:
        iface = conf.iface
    print("Capturing on '%s'" % iface)

    # This should be a nonlocal variable, using a mutable object
    # for Python 2 compatibility
    i = [0]

    def _cb(pkt):
        # type: (Packet) -> None
        print("%5d\t%s" % (i[0], pkt.summary()))
        i[0] += 1

    sniff(prn=_cb, store=False, *args, **kargs)
    print("\n%d packet%s captured" % (i[0], 's' if i[0] > 1 else ''))
```

Додаток Б

Блок схема процесу ініціалізації засобу та запуску графічної обгортки

Додаток Г

Схема надсилання моніторинг статистики на сервер

Додаток Ґ

Код головоного файлу запуску засобу

```
import threading

from tkinter import ttk, CENTER
import customtkinter as ctk
from .CTkDataVisualizingWidgets import CTkChart
from scapy.all import sniff
from PIL import Image

from gui.package_frame import PackageInfo

ctk.set_default_color_theme("green")
ctk.set_appearance_mode("green")

WIDTH = 1100
HEIGHT = 450
custom_font = ("Noto Sans Mono CJK KR", 20)


class Sniff(ctk.CTkFrame):

    def __init__(self, root, **kwargs):
        super().__init__(root, **kwargs)
        self.counter = 0
        self.back_button = ctk.CTkButton(self, text="Back", text_color="black",
fg_color="#51CC3D", font=custom_font,
                                         corner_radius=0)
        self.refresh_button = ctk.CTkButton(self, text="Refresh",
text_color="black", fg_color="#51CC3D",
                                            font=custom_font,
                                            corner_radius=0, command=self.refresh)
        self.stop_button = ctk.CTkButton(self, text="Stop", text_color="black",
fg_color="#51CC3D", font=custom_font,
                                         corner_radius=0)

        self.info_frame = ctk.CTkFrame(self, width=WIDTH, height=320, )
        self.tree = ttk.Treeview(self.info_frame, show='headings', height=20)
        self.tree['columns'] = ("No.", 'Source', 'Destination', 'Protocol', 'Info')

        self.tree.column("No.", anchor=CENTER)
        self.tree.column("Source", anchor=CENTER)
        self.tree.column("Destination", anchor=CENTER)
        self.tree.column("Protocol", anchor=CENTER)
        self.tree.column("Info", anchor=CENTER, width=300)

        self.tree.heading("No.", text="No.")
        self.tree.heading("Source", text="Source")
```

Продовження додатку Г

```
        self.tree.heading("Destination", text="Destination")
        self.tree.heading("Protocol", text="Protocol")
        self.tree.heading("Info", text="Info")

        vsb = ttk.Scrollbar(self.info_frame, orient="vertical",
command=self.tree.yview)
        vsb.pack(side='right', fill='y')

        self.tree.configure(yscrollcommand=vsb.set)

        self.back_button.grid(row=0, column=0, padx=45, pady=5, ipadx=50, ipady=5,
sticky="nsew")
        self.refresh_button.grid(row=0, column=1, padx=45, pady=5, ipadx=50,
ipady=5, sticky="nsew")
        self.stop_button.grid(row=0, column=2, padx=45, pady=5, ipadx=50, ipady=5,
sticky="nsew")

        self.info_frame.grid(row=2, rowspan=3, columnspan=3, sticky="wes")

        self.sniffing_option()

    def packet_callback(self, packet):
        packet_info = PackageInfo(packet=packet, counter=self.counter).get()
        self.tree.insert("", "end", values=packet_info)
        self.counter += 1

    def sniffing_option(self):
        self.sniff_thread = threading.Thread(target=self.start_sniffing)
        self.sniff_thread.start()
        self.tree.pack()

    def refresh(self):
        for i in self.tree.get_children():
            self.tree.delete(i)
        self.counter = 0

    def start_sniffing(self):
        sniff(prn=self.packet_callback, filter='ip', store=0)


class Menu(ctk.CTkFrame):
    def __init__(self, root, **kwargs):
        super().__init__(root, **kwargs)

        self.analyzer_button = ctk.CTkButton(self, text="Analyzer",
fg_color="#51CC3D", text_color="black",
                                             font=custom_font,
```

Продовження додатку Ґ

```
                                                corner_radius=0,
command=self.analyze_button_action)
        self.map_button = ctk.CTkButton(self, text="Map", text_color="black",
fg_color="#51CC3D", font=custom_font,
                                        corner_radius=0,
command=self.map_button_action)
        self.log_button = ctk.CTkButton(self, text="Logs", text_color="black",
fg_color="#51CC3D", font=custom_font,
                                        corner_radius=0,
command=self.log_button_action)
        self.image =
ctk.CTkImage(light_image=Image.open("/home/user/my_projects/PA/gui/logo.png"),
size=(500, 400))
        self.label = ctk.CTkLabel(self, text="", image=self.image)

        self.grid_rowconfigure(1, weight=1, minsize=10)
        self.grid_rowconfigure(5, weight=1, minsize=10)
        self.grid_columnconfigure(1, weight=1)
        self.grid_columnconfigure(3, weight=1)

        self.analyzer_button.grid(row=2, column=0, padx=20, pady=5, ipadx=50,
ipady=5, sticky="nsew")
        self.map_button.grid(row=3, column=0, padx=20, pady=5, ipadx=50, ipady=5,
sticky="nsew")
        self.log_button.grid(row=4, column=0, padx=20, pady=5, ipadx=50, ipady=5,
sticky="nsew")
        self.label.grid(row=1, column=2, rowspan=5, padx=20, pady=5, sticky="e")

    def analyze_button_action(self):
        self.forget()
        sniff_frame = Sniff(self.master, height=HEIGHT, width=WIDTH,
fg_color="Black")
        sniff_frame.pack(expand=True, fill="both")

    def map_button_action(self):
        self.forget()

        graph_frame = ctk.CTkFrame(self.master, height=HEIGHT, width=WIDTH,
fg_color="Black")
        value = {'Spotify': 5, 'YouTube': 25, 'Linkedin': 7, 'store': 3,
'Realpython': 15, 'Github': 10, 'Outlook': 1, 'dev': 12, "gmail": 4}
        CTkChart(graph_frame, value, corner_radius=5,
                 show_indicators=(True, True), stat_info_show=(True, True),
chart_arrow="none", border_width=20,
                 stat_width=50,
                chart_axis_width=20, width=WIDTH, height=HEIGHT).pack(side="left",
pady=20,
```

Продовження додатку Ґ

```
         padx=20)

        graph_frame.pack(expand=True, fill="both")

    def log_button_action(self):
        print("log")


class App(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.geometry(f"{WIDTH}x{HEIGHT}")
        self.title("PA")
        self.minsize(750, 500)

        bg_color =
self._apply_appearance_mode(ctk.ThemeManager.theme["CTkFrame"]["fg_color"])
        text_color =
self._apply_appearance_mode(ctk.ThemeManager.theme["CTkLabel"]["text_color"])
        selected_color =
self._apply_appearance_mode(ctk.ThemeManager.theme["CTkButton"]["fg_color"])

        treestyle = ttk.Style()
        treestyle.theme_use('default')
        treestyle.configure("Treeview", background=bg_color, foreground=text_color,
fieldbackground=bg_color,
                            borderwidth=0)
        treestyle.map('Treeview', background=[('selected', bg_color)],
foreground=[('selected', selected_color)])
        self.bind("<<TreeviewSelect>>", lambda event: self.focus_set())

        self.menu_frame = Menu(self, height=HEIGHT, width=WIDTH, fg_color="Black")
        self.menu_frame.pack(expand=True, fill="both")


def main_window():
    app = App()
    app.mainloop()


def run():
    main_window()
```