ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Впровадження дистанційного управління сервером Wireguard з використанням системи Ansible»

на здобуття освітнього ступеня магістра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Андрій НЕДАВНІЙ
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав:
здобувач вищої освіти Андрій НЕДАВНІЙ
група КСДМ-61

Керівник: Андрій ЛЕМЕШКО
науковий ступінь, вчене звання доктор філософії, доцент

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти Магістр
Спеціальність 123 Комп'ютерна інженерія
Освітньо-професійна програма Комп'ютерні системи та мережі

ЗАТВЕРДЖУЮ
Завідувач кафедрою _____
_____
«____»_____20__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Недавньому Андрію Володимировичу
(прізвище, ім'я, по батькові здобувача)

1.Тема кваліфікаційної роботи: «Впровадження дистанційного управління сервером Wireguard з використанням системи Ansible»

керівник кваліфікаційної роботи: Андрій ЛЕМЕШКО, доктор філософії, доцент,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «__»__.20__ р. № __

2. Строк подання кваліфікаційної роботи «__» __ 202__ р.

3. Вихідні дані до кваліфікаційної роботи: Віддалене адміністрування, сервер, платформа Unix, система управління конфігураціями Ansible, віртуальні приватні мережі, Wireguard

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1. Аналіз проблеми; Актуалізація автоматизації рутинних задач в роботі адміністратора мережі.
2. Методика дослідження шляхів вирішення проблеми; Аналіз та дослідження основ функціонування систем управління конфігураціями, Wireguard та операційних систем на платформі Unix.
3. Результати дослідження; Приклади роботи системи управління конфігураціями Ansible в операційній системі Debian.

4. Висновки. Аналіз отриманого в результаті роботи матеріалу, огляд можливостей для подальшого використання.
5. Перелік графічного матеріалу: *презентація*
6. Дата видачі завдання «__»__.20__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури		
2	Вивчення матеріалів для подальшого аналізу		
3	Аналіз Ansible та unix-based операційних систем		
4	Аналіз особливостей протоколу Wireguard		
5	Дослідження можливостей застосування Ansible		
6	Впровадження віддаленого управління сервером		
7	Оформлення роботи: вступ, висновки, реферат		
8	Розробка демонстраційних матеріалів		

Здобувач вищої освіти _____ Андрій НЕДАВНІЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Андрій ЛЕМЕШКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня магістра

Направляється здобувач Недавній А.В. до захисту кваліфікаційної роботи
за спеціальністю 123 Комп'ютерна інженерія
освітньо-професійної програми Комп'ютерні системи та мережі
на тему: «Впровадження дистанційного управління сервером Wireguard з використанням системи Ansible».
Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ _____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Недавній А.В. під час виконання кваліфікаційної роботи показав хорошу теоретичну та практичну підготовку, вміння користуватись науковою та технічною літературою, продемонстрував відповідальне ставлення до роботи. За формою і змістом кваліфікаційна робота відповідає чинним вимогам, є самостійною роботою, у якій студент показав знання засад спеціальності, знання щодо конкретного предмету своєї роботи, вміння отримувати інформацію за допомогою сучасних наукових методів, вміння осмислювати отриману інформацію і подавати її в прийнятній для даної галузі знань формі.

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача Недавнього А.В. на оцінку «_____» та присвоїти йому кваліфікацію _____.

Керівник кваліфікаційної роботи _____ _____
(підпис) *(Ім'я,ПРІЗВИЩЕ)*
«_____»_____ 202_ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач Недавній А.В. допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою _____ _____ _____
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

«_____»_____ 202_ року

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну магістерську роботу

здобувача вищої освіти Недавнього Андрія Володимировича
на тему: «Впровадження дистанційного управління сервером Wireguard з використанням системи Ansible».

Актуальність.

В наш час життя без комп'ютерів майже неможливе. Комп'ютери необхідні вдома, на виробництві, в банківській справі, в кіно-індустрії і ще багато-багато де. Вони відкрили якісно новий етап в житті і розвитку людської цивілізації. Для ефективного керування та розгортання комп'ютерних мереж необхідною є участь фахівців з комп'ютерної інженерії – системних адміністраторів та системних програмістів. Дуже важливим, вичерпним та невідновлюваним ресурсом є час, який витрачають фахівці для реалізації тих чи інших технічних або програмних рішень. Для оптимізації використання робочого часу та автоматизації роботи являється доцільним використання програмного забезпечення, приклад якого було приведено в роботі, що дозволить суттєво зменшити витрати часу на однотипні задачі. Тому, кваліфікаційна робота на тему "Впровадження дистанційного управління сервером Wireguard з використанням системи Ansible " є актуальною.

Позитивні сторони.

1. Автор роботи проаналізував актуальну на сьогоднішній час операційну платформу;
2. В роботі були розглянуті способи використання системи управління конфігураціями;
3. Приведено приклади роботи в реальному середовищі з використанням модулів, які використовуються в роботі мережевого адміністратора найчастіше;
4. В роботі використовується відносно новий та актуальний протокол Wireguard;
5. Доволі детальний опис роботи протоколу.

Недоліки.

1. Поверхневий аналіз функціональних основ Unix систем;
2. Стислий огляд основних функцій та модулів системи управління конфігураціями Ansible.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної магістерської роботи.

Висновок: *кваліфікаційна магістерська робота заслуговує оцінку "_____", а здобувач _____ заслуговує присвоєння кваліфікації: _____*

Рецензент: _____ _____
науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 84 стор., 50 рис., 11 джерел.

Мета роботи – дослідити функції сервера Wireguard і впровадити віддалене налаштування використовуючи систему управління конфігураціями Ansible.

Об'єкт дослідження – сервер Wireguard.

Предмет дослідження – впровадження віддаленого налаштування за допомогою системи управління конфігураціями Ansible для сервера Wireguard.

Короткий зміст роботи: Визначено, що використання систем управління конфігураціями дозволяє значно підвищити продуктивність роботи адміністраторів систем і мереж, заощаджуючи час на виконання рутинних завдань. Автоматизовані процеси допомагають уникнути помилок, забезпечуючи виконання завдань через код. Такі інструменти можуть бути успішно впроваджені в різних галузях підприємництва та бізнесу, які мають власні IT-підрозділи для обслуговування мережевої інфраструктури.

Здійснено аналіз структури, функцій та основи роботи серверу Wireguard. Також були розглянуті приклади використання Wireguard в операційній системі Debian та Windows. Поверхнево розглянуто структуру, основні характеристики операційних систем на платформі Unix.

КЛЮЧОВІ СЛОВА: ВІДДАЛЕНЕ АДМІНІСТРУВАННЯ, СЕРВЕР, ПЛАТФОРМА UNIX, СИСТЕМА УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ANSIBLE, WIREGUARD

ABSTRACT

Text part of the master's qualification work:

84 pages, 50 pictures, 11 sources.

The purpose of the work to study the functions of the Wireguard server and implement remote configuration using the Ansible configuration management system.

Object of research – the Wireguard server.

Subject of research – implementation of remote configuration using the Ansible configuration management system for the Wireguard server.

Summary of the work: It has been determined that the use of configuration management systems can significantly increase the productivity of system and network administrators, saving time on routine tasks. Automated processes help to avoid errors by ensuring that tasks are performed through code. Such tools can be successfully implemented in various industries and businesses that have their own IT departments to maintain network infrastructure.

The article analyzes the structure, functions, and basics of the Wireguard server. Examples of using Wireguard in the Debian and Windows operating systems were also considered. The structure and main characteristics of operating systems on the Unix platform are superficially considered.

KEYWORDS: REMOTE ADMINISTRATION, SERVER, UNIX PLATFORM, ANSIBLE CONFIGURATION MANAGEMENT SYSTEM, WIREGUARD

ЗМІСТ

ВСТУП

За останні десятиліття інформаційні технології різко змінили ландшафт бізнесу та суспільства, перетворивши його у глобальну мережу, де забезпечення безпеки і приватності даних стало основоположним аспектом. Разом із зростанням розподілених робочих місць, віддалених команд і збільшенням кількості переносних пристроїв, виникла необхідність у створенні надійних та ефективних механізмів для забезпечення безпеки мережевого зв'язку.

Одним із відповідей на цей виклик стала технологія віртуальних приватних мереж (VPN). Сервер Wireguard, який представляє собою швидкий та збалансований VPN-сервер, вирізняється ефективністю, простотою реалізації та надійністю. Його простий у використанні протокол, який базується на сучасних криптографічних протоколах, дозволяє створювати безпечні тунелі зв'язку між різними мережевими вузлами.

Однак, при впровадженні VPN-серверів і клієнтських з'єднань стикаються з викликами, такими як налаштування, управління та підтримка цих систем. Ручне налаштування та управління може бути часозатратним та призводити до помилок, особливо при великих обсягах.

Ця магістерська робота присвячена дослідженню системи управління конфігураціями Ansible та впровадженню віддаленого налаштування сервера Wireguard з її використанням. Головна мета полягає в дослідженні можливостей технології Wireguard, оцінці її продуктивності та безпеки, а також розробці Ansible-плейбуків для автоматизації налаштування та управління. Основні завдання роботи включають аналіз можливостей системи Ansible, розробку зразка інфраструктури з використанням Wireguard та Ansible.

1 АНАЛІЗ СИСТЕМ УПРАВЛІННЯ КОНФІГУРАЦІЯМИ

1.1 Огляд систем управління конфігураціями

Управління конфігураціями - це систематичний підхід до контролю та обробки змін у продукті з метою збереження його цілісності і стабільності. Цей термін виник у сфері програмування, але зараз широко використовується в контексті управління серверами.

В автоматизації та пов'язаних з нею інструментах закладено ключовий роль у конфігураційному управлінні. Це дозволяє серверам досягати запланованого стану, використовуючи конкретну мову, інструмент чи функції. Необхідність автоматизації стала найбільш критичною складовою управління конфігураціями серверів.

На сьогоднішній день існує безліч інструментів для управління конфігураціями, серед яких найпопулярнішими є Puppet, Ansible, Chef і Salt. Кожен із цих інструментів має свої особливості та переваги, проте всі вони спрямовані на одну головну мету - забезпечення відповідності системи до стану, описаного в заданих сценаріях.

Управління конфігураціями має ряд основних функцій, які сприяють контролю та забезпечують цілісність IT-інфраструктури та її елементів:

Збір інформації: Функція полягає у систематичному зборі інформації про кожен конфігураційний елемент і компонент IT-інфраструктури.

Визначення та аналіз зв'язків: Проведення аналізу зв'язків та взаємодій між різними конфігураційними елементами для кращого розуміння системи.

Контроль цілісності: Після кожної зміни в конфігурації серверів здійснюється контроль, щоб забезпечити правильність та стабільність системи.

Моніторинг та аналіз: Постійне стеження за IT-інфраструктурою та її аналіз для виявлення потенційних проблем або несанкціонованих змін.

Що стосується інструментів для управління конфігураціями серверів, вони надають такі важливі можливості:

Автоматизація: Кожен інструмент має свій власний синтаксис та набір функцій для створення сценаріїв автоматизації. Це дозволяє забезпечити автоматичний розгортання та налаштування серверів.

Ідемпотентність: Системи управління конфігурацією слідкують за станом ресурсів, щоб уникнути повторення завдань, які вже виконані. Це допомагає забезпечити стабільність і послідовність дій.

Детальна інформація про систему: Інструменти надають докладну інформацію про систему, зокрема мережеві інтерфейси, IP-адреси, операційні системи тощо. Це полегшує створення універсальних та адаптивних сценаріїв.

Система шаблонів: Більшість інструментів мають вбудовані шаблони, які дозволяють швидко створювати конфігураційні файли та сервіси з можливістю використання змінних та умовних виразів.

Розширюваність: Сценарії для керування конфігураціями можна індивідуалізувати та адаптувати під потреби конкретного сервера, а також повторно використовувати фрагменти сценаріїв.

Ці функції та можливості інструментів допомагають ефективно керувати конфігураціями серверів і забезпечити стабільну та безперебійну роботу інфраструктури.

Система управління конфігураціями - це програмне забезпечення, яке забезпечує централізоване керування багаточисельними операційними системами та програмами, що працюють на них. Сучасні системи управління конфігураціями прагнуть реалізувати концепцію "Інфраструктура як код", де інфраструктура центрів обробки даних описується через файли конфігурації, що дозволяє уникнути ручного редагування файлів на серверах.

Існують різні підходи до управління конфігураціями:

Функціональний: Сконцентрований на описі того, як повинна виглядати цільова конфігурація.

Процедурний: Зосереджений на змінах, які потрібно внести в конфігурації.

Інтелектуальний: Описує причини, з яких інфраструктура повинна бути налаштована саме таким чином.

Розглянемо сценарій, коли необхідно керувати великим мультивендорним парком серверів, розташованих у різних країнах. Ручне або скриптове керування кожним з цих серверів може бути затратним і призводити до помилок.

Оптимальним рішенням у даному випадку є використання системи віддаленого управління конфігураціями, такої як Ansible. Використання такої системи дозволить описати потрібний кінцевий стан хоста без необхідності вручну змінювати або підлаштовувати скрипти. Ansible дозволить ефективно управляти конфігураціями серверів та забезпечити стабільну та надійну роботу інфраструктури.

1.2 Основи Ansible

Ansible – це потужна система управління конфігураціями, що надає можливість автоматизувати та спростити налаштування, обслуговування та управління серверами та службами. Застосовуючи модулі, які входять до складу Ansible або створюючи свої власні, можна виконувати різноманітні дії на віддалених хостах.

З більш ніж 200 модулями, які поставляються з Ansible, ця система має широкий функціонал та продовжує розвиватись з кожною новою версією. Використання Ansible не потребує встановлення агента на хостах, що робить його зручним для роботи з мережевим обладнанням. Крім того, простий інтерфейс та багато інформації на офіційному форумі дозволяють легко почати працювати з цією системою.

Ansible дозволяє створювати сценарії (playbooks), в яких описується бажаний стан хостів, і автоматично впроваджує необхідні зміни для досягнення цього стану. Це спрощує і прискорює процес переконфігурації системи.

Основні задачі, які можна виконувати за допомогою Ansible, включають підключення по SSH до пристроїв, відправлення команд на хости, групування хостів і відправлення команд окремим групам, а також створення користувацьких ролей для полегшення конфігурації та багаторазового використання коду.

Ansible є ідемпотентним, оскільки він відстежує стан системних ресурсів на керованих хостах, щоб уникнути повторення завдань, які вже були виконані. Це забезпечує стабільність та надійність виконання сценаріїв, а також дозволяє досягати бажаного стану системи навіть при повторних запусках. Ansible є незамінним інструментом для автоматизації управління конфігураціями та забезпечення ефективної роботи з віддаленими серверами.

1.3 Конфігураційний файл Ansible

Файл конфігурації Ansible може знаходитись в різних розташуваннях залежно від місця, де встановлено пакет Ansible. Зазвичай за замовчуванням цей файл розташовується у шляху /etc/ansible/ansible.cfg.

Зазвичай він має формат YAML, що полегшує читання та зміну. Основні розділи файлу визначають різні параметри, такі як налаштування підключення до серверів, шляхи до ключів SSH, конфігурації модулів та інші опції. Одним із ключових аспектів конфігурації є визначення хостів, з якими Ansible буде взаємодіяти. Це може бути визначено в розділі "inventory", де перелічені сервери або групи серверів, що мають певні ролі в інфраструктурі. Кожен хост може мати свої унікальні параметри, які визначаються в цьому файлі конфігурації. Також, в файлі конфігурації Ansible можна знайти налаштування параметрів підключення до серверів. Це включає в себе такі аспекти, як тип підключення (SSH, наприклад), порт, ім'я користувача, аутентифікаційні ключі та інші параметри, необхідні для ефективного обміну даними між контролером Ansible та цільовими серверами. Попередньо налаштовані плагіни та модулі також можуть бути визначені у файлі конфігурації, дозволяючи Ansible розпізнавати та використовувати різноманітні розширення для спрощення конфігураційних задач. Окрім того, файл конфігурації Ansible визначає правила та політики обробки помилок, а також виведення результатів виконання завдань. Це становить важливу частину ефективного використання Ansible в реальних сценаріях автоматизації.

У файлі конфігурації Ansible доступно безліч параметрів, які дозволяють налаштовувати його поведінку. Розглянемо лише деякі з них:

Секція [defaults]: В цій секції вказуються загальні параметри конфігурації за замовчуванням.

Параметр inventory: Цей параметр вказує шлях до файлу інвентаря (inventory), в якому зберігається список хостів та груп, з якими Ansible буде працювати.

Параметр module_utils: Вказує шлях до директорії, де знаходяться користувацькі модулі (module_utils).

Параметр sudo_user: Вказує під яким користувачем Ansible буде виконувати команди з підвищеними привілегіями (наприклад, sudo).

Параметр remote_port: Вказує на який порт SSH встановлювати з’єднання з віддаленими хостами.

Параметр host_key_checking: Цей параметр визначає, чи слід виконувати перевірку ключів хостів при SSH-з’єднанні. Значення "False" вимикає перевірку, що може бути корисно при автоматизованому розгортанні.

Параметр roles_path: Вказує шлях до директорії, де знаходяться ролі (roles). Ролі є перевикористовуваними наборами задач і папок, які можуть використовуватись у різних проектах.

Застосування цих параметрів дозволяє налаштовувати Ansible згідно з потребами проекту та забезпечити ефективне та автоматизоване управління конфігураціями.

1.4 Файл інвентарю в Ansible

Інвентарний файл – це файл, який містить інформацію про хости та групи хостів, до яких будуть застосовуватись команди, модулі та завдання Ansible.

У цьому файлі можна вказати хости окремо, по одному або згруповано за допомогою IP-адрес або DNS-імен. Такий підхід дозволяє організувати групи пристроїв з певними спільними характеристиками, наприклад, операційною системою або ролями, і застосовувати до них однакові конфігураційні завдання. Наприклад, якщо у вас є кілька центрів обробки даних, то ви можете створити

групи Ansible для комутаторів, що потребують спільного набору операцій, таких як оновлення операційної системи та перезавантаження пристроїв.

Формат інвентарного файлу може бути ini або yaml. За замовчуванням, файл інвентаря розташовується у шляху /etc/ansible/hosts.

Ось приклад вмісту інвентарного файлу:

Приклад файлу:

```
[example]
host1 ansible_host=10.1.1.1
[example:vars]
ansible_user=cessabit
ansible_password=P@ssw0rd!
ansible_ssh_private_key_file=/home/cessabit/ownCloud/ssh/cessabit
```

Рисунок 1.1 – Приклад файлу hosts

де [example] назва групи;

host1 ansible_host=10.1.1.1 назва хоста в Ansible та IP-адрес;

[example:vars] змінні для групи;

ansible_user=cessabit вказує під яким користувачем логінитись на хост;

ansible_password=P@ssw0rd! пароль для користувача під яким логіниться;

ansible_ssh_private_key_file=/home/cessabit/ownCloud/ssh/cessabit місце розташування особистого приватного ключа ssh.

За замовчуванням існує дві групи all та ungrouped. Перша включає в себе всі хости, а друга тільки хости які не входят ні в одну із груп.

1.5 Playbook: створення та виконання

Playbook (файл сценарія) – файл в якому описуються дії, які потрібно виконати на певній групі хостів або на хості. Ділиться на play(набір задач) та tasks(конкретні задачі), в свою чергу, tasks ділиться на pre_tasks(виконуються перед основним таском) та post_tasks(виконуються після основного таска).

Всі сценарії Ansible пишуться на YAML.

YAML – формат файлу, який нагадує JSON, але набагато легше для сприйняття людиною. Основні поняття YAML, які найбільше потрібні при написанні сценарії:

Початок файлу YAML починається з трьох дефісів. Але Ansible не буде вважати помилкою, якщо ви за будете указати три дефіса на початку сценарію. Коментарі починаються зі знаку решітки і продовжуються до кінця рядку, як в сценаріях на мові командної оболонки. Зазвичай рядки YAML не знаходяться у лапках. Хоча це не забороняється. Іноді Ansible вимагає укладати рядки в лапки. Зазвичай це рядки з фігурними дужками { }, які використовуються для встановлення значень змінних. Під час написання сценаріїв часто виникають ситуації, коли необхідно передати модулю багато аргументів. В естетичних цілях їх можна розмістити у кілька рядків у файлі. Однак при цьому необхідно, щоб Ansibie сприймав їх як єдиний рядок. У YAML для цього можна скористатися знаком більше (>). Парсер YAML у цьому випадку замінить розриви рядків пробілами. YAML має власний логічний тип. Він пропонує широкий вибір рядків, які можуть інтерпретуватися як «істина» та «брехня».

Кожний сценарій повинен містити:

– список хостів які потрібно налаштувати;
– список задач, які потрібно виконати на цих хостах.
– окрім хостів і задач, сценарії також повинні мати параметри. Основними параметрами являються:
– name;
– become;
– vars.

name – коментарій, який описує задачу. Ansible виводить його перед стартом. Дуже полегшує дебаг сценаріїв.

become – якщо має значення yes, Ansible буде виконувати всі задачі з правами суперкористувача. Це стає корисним при управління серверами на ОС Ubuntu, тому що данна система забороняє виконувати операції із системними файлами без прав суперкористувача.

vars – список перемінних та їх значень.

Приклад файлу:

```
hosts: vpntermun53
become: yes
vars:
  wireguard_conf: /etc/wireguard/server.conf
tasks:
  - name: install package wireguard
    yum:
      name: wireguard
      state: latest
  - name: add ipv4.ip_forwarding
    run_once: true
    lineinfile:
      state: present
      path: /etc/sysctl.conf
      line: "net.ipv4.ip_forward=1"
  - name: check server.conf file
    shell: cat {{ wireguard_conf }}
    register: result
  - name: start and enabled wireguard
    systemctl:
     name: wg-quick@server
      state: started
      enabled: yes
  - debug:
      var: result.stdout_lines
```

Рисунок 1.2 – Приклад файлу сценарію

Для запуску playbooks використовується команда:

ansible-playbook <шлях до playbook>

Коли ви запускаєте команду ansible-playbook, вона виводить інформацію про стан кожного завдання, яке виконується в рамках операції. Зверніть увагу, що стан одних завдань може бути вказано як changed, а інших – ок. Будь-яке запущене завдання потенційно може змінити стан хоста. Перед тим, як зробити будь-яку дію, модулі перевіряють, чи потрібно змінити стан хоста. Якщо стан хоста відповідає значенням аргументів модуля, Ansible не робить жодних дій і повідомляє, що статус ок. Якщо між станом хоста та значеннями аргументів модуля є різниця, Ansible вносить зміни до стану хоста та повідомляє, що статус був змінений (changed).

1.6 Використання ролей в Ansible

Ролі в Ansible - це набори незалежних компонентів, які дозволяють повторно використовувати загальні кроки конфігурації та автоматизувати рутинні задачі. Вони спрощують написання сценаріїв та дозволяють з легкістю використовувати їх на різних хостах. Ролі організовані за допомогою попередньо визначеної структури каталогів.

Кожна роль має унікальну назву, наприклад, "wireguard-role". Структура каталогів ролі включає наступні директорії:

defaults: містить змінні за замовчуванням для ролі. Ці змінні мають найнижчий пріоритет і можуть бути легко змінені.

vars: містить змінні для ролі. Змінні в цій директорії мають вищий пріоритет, ніж змінні з директорії defaults.

tasks: містить основний список задач, які має виконати роль.

files: містить файли, які будуть скопійовані на віддалені хости. Шлях до ресурсів не потрібно вказувати.

templates: містить шаблони файлів, що підтримують зміни з ролі. Для створення шаблонів використовується мова шаблонів Jinja2.

meta: містить метадані ролі, такі як автор, підтримувані платформи, залежності.

handlers: містить обробники, які можуть бути викликані директивою notify та пов'язані з сервісом.

Для використання ролі, потрібно створити стандартний playbook та вказати в ньому потрібні ролі для встановлення. Завдяки ролям, можна легко структурувати та перевикористовувати код, що значно спрощує процес автоматизації та управління конфігурацією серверів і мережевого обладнання за допомогою Ansible.

Приклад такого файлу:

```
---
- hosts: vpntermun53
  roles:
    - wireguard-role
    - rsyslog-role
    - iptables-default-role
    - node-exporter-role
…
```

Рисунок 1.3 – Приклад файлу сценарії для виконання ролей

За замовчуванням Ansible знаходить ролі в двох основних місцях: в каталозі "role" на тому ж рівні, де знаходиться наш файл playbook, і в системному каталозі "/etc/ansible/roles". Однак, у вас є можливість зберігати свої ролі в інших каталогах, якщо задати параметр конфігурації "roles_path" в файлі ansible.cfg.

Наприклад, ви можете вказати шлях "/home/cessabit/ownCloud/roles/" для збереження своїх ролей.

Крім того, Ansible має інтегрований репозиторій ролей під назвою Ansible Galaxy. Це загальнодоступний репозиторій, де спільнота може публікувати та ділитися своїми ролями. Ansible Galaxy служить як громадський репозиторій, де ви знайдете велику кількість різноманітних ролей для різних завдань.

Для взаємодії з Ansible Galaxy ви можете використовувати утиліту командного рядка Ansible Galaxy CLI. Ось деякі приклади використання команд:

ansible-galaxy search wireguard: Ця команда допоможе знайти ролі, пов'язані з WireGuard, на Ansible Galaxy.

ansible-galaxy info wireguard: Ця команда показує інформацію про певну роль, наприклад, "wireguard", знайдену на Ansible Galaxy.

ansible-galaxy install wireguard: Ця команда встановлює певну роль, наприклад, "wireguard", із репозиторію Ansible Galaxy, щоб ви могли використовувати її у своїх сценаріях.

Ansible Galaxy спрощує процес знаходження, спільного використання та використання ролей, що допомагає прискорити розробку та автоматизацію процесів конфігурації.

2 АНАЛІЗ UNIX-BASED СИСТЕМ

2.1 Сучасні версії ОС Unix

Операційна система є комплексом взаємопов'язаних системних програм, які забезпечують контроль використання та розподіл ресурсів комп'ютерної системи, а також забезпечують взаємодію користувача з комп'ютером. Сімейство сучасних Unix-подібних систем представляє різноманітну групу операційних систем, таких як System V, BSD та GNU/Linux. Операційні системи Unix працюють на комп'ютерах з різноманітною архітектурою.

Linux, з'явившись середині 1990-х років, швидко здобув широку користувацьку базу, охопивши практично всі куточки світу. Ця операційна система присутня в телефонах, термостатах, автомобілях, холодильниках та телевізорах. Вона також використовується для управління більшістю Інтернету, найпотужнішими суперкомп'ютерами та світовими фондовими біржами.

В наш час, різні операційні системи конкурують за першість на ринку персональних комп'ютерів. Впевнено можна сказати, що ОС Windows є найпопулярнішою серед домашніх користувачів, оскільки вона орієнтована на зручне графічне інтерфейсне управління, що дозволяє навіть не досвідченим користувачам легко користуватись нею. З іншого боку, Linux розробляється з упором на командний рядок (CLI), що може створювати певні бар'єри для новачків. Хоча Linux також має графічний інтерфейс у більшості дистрибутивів, його потужності особливо проявляються на серверних платформах завдяки відсутності візуалізації, що забезпечує економію ресурсів і підвищує продуктивність та надійність.

У процесі розвитку Linux з'явилося багато різних реалізацій цієї операційної системи у вигляді різних дистрибутивів. Кожен з дистрибутивів має свої особливі корисні можливості, відповідно до поглядів та потреб розробників. Згодом з'явилася необхідність у створенні уніфікованої реалізації, яка об'єднала багато важливих нововведень і вдосконалень.

Серед деяких основних Unix-based ОС варто зазначити наступні:

MacOS – операційна система розроблена компанією Apple і є однією з найпопулярніших у сучасному світі. Вона спеціально оптимізована для пристроїв лінійки Mac і має свої унікальні функції. MacOS заснована на ядрі Darwin, яке в свою чергу виходить з декількох різних версій Unix. Вона поєднує класичний Unix з інноваціями, що додані розробниками Apple. MacOS має користувацький інтерфейс Aqua, що забезпечує просте та ефективне керування системою. Проте, вона є комерційною розробкою з закритим вихідним кодом.

Debian - це один з найдавніших і популярних дистрибутивів GNU/Linux, який заснований на ядрі Linux. Його розроблена відкрита спільнота і він представляє собою безкоштовне програмне забезпечення з відкритим кодом. Debian знаменитий своєю стабільністю, надійністю та гнучкістю. Він має велику базу пакетів та дозволяє легко налаштовувати систему під індивідуальні потреби. Debian став основою для багатьох інших дистрибутивів, включаючи Ubuntu.

Ubuntu - це популярний дистрибутив GNU/Linux, заснований на Debian. Розроблений і підтримується компанією Canonical, Ubuntu має активну спільноту та забезпечує легкий доступ до великої кількості безкоштовного програмного забезпечення. Його філософія полягає в тому, щоб зробити Linux доступним та зручним для звичайних користувачів. Він надає простий інтерфейс, швидку інсталяцію, а також можливість оновлення системи зручними засобами.

Linux Mint - це інший дистрибутив GNU/Linux, який також базується на Debian та Ubuntu. Він вирізняється своєю простотою використання та ефектним дизайном. Linux Mint прагне забезпечити зручну і підтримувану операційну систему для користувачів будь-якого рівня досвіду. Цей дистрибутив активно розвивається спільнотою та пропонує багато корисних програм і функцій.

Описані дистрибутиви займають значне місце в ринку операційних систем та забезпечують високий рівень функціональності та відповідь на потреби різних користувачів. Кожен з них має свої унікальні риси і можливості, що дозволяє обирати оптимальний варіант в залежності від потреб і вимог.

Linux є операційною системою, яка використовується в різних платформах, включаючи популярну мобільну платформу Android. Операційна система є

необхідною для організації керування апаратними ресурсами комп'ютера та забезпечення зв'язку між програмним забезпеченням і апаратурою.

Операційна система Linux складається з кількох важливих компонентів, які спільно забезпечують її функціонування:

Завантажувач: Це програмне забезпечення, яке керує процесом завантаження комп'ютера. Для користувачів, це зазвичай просто заставка, яка з'являється під час запуску системи.

Ядро: Це єдина частина, що насправді називається "Linux". Ядро керує процесором, пам'яттю та периферійними пристроями, і воно є найнижчим рівнем операційної системи.

Система ініціалізації: Це підсистема, яка завантажує користувацький простір і керує демонами - фоновими службами, які працюють під час завантаження або після входу на робочий стіл.

Демони: Це фонові служби, такі як друк, звук, планування та інші, які автоматично запускаються під час завантаження системи або після входу користувача.

Графічний сервер: Ця підсистема відповідає за відображення графічного інтерфейсу на моніторі. Відомий як X-сервер або просто X.

Середовище робочого столу: Це частина операційної системи, з якою користувачі фактично взаємодіють. Існує багато варіантів середовищ робочого столу, таких як GNOME, KDE, Xfce та інші, кожне з яких має свої унікальні риси та додаткові програми.

Додаткові програми: Окрім середовищ робочого столу, Linux пропонує багато програм, які можна встановити для різних потреб користувача. Більшість дистрибутивів Linux мають вбудовані центри програм, що дозволяють швидко знаходити та встановлювати необхідне програмне забезпечення.

Ці компоненти працюють спільно, забезпечуючи зручність, надійність та ефективність операційної системи Linux. Кожен з них відповідає за певний аспект функціонування системи, що дозволяє їй працювати гармонійно і задовольняти потреби різних користувачів.

Крім основних компонентів, операційна система Linux також має багато додаткових складових, що дозволяють забезпечувати різноманітні функціональні можливості та покращувати її продуктивність. Ось кілька з них:

Пакетний менеджер: Кожна Linux-дистрибуція має свою систему керування пакетами. Вона дозволяє легко встановлювати, оновлювати та видаляти програми і пакети з великої колекції програмного забезпечення. Наприклад, Ubuntu використовує APT (Advanced Packaging Tool), а Fedora використовує DNF (Dandified YUM).

Термінал: Linux надає доступ до командного рядка або терміналу, який дозволяє користувачеві взаємодіяти з системою через текстовий інтерфейс. Це дає користувачам більше контролю над операційною системою та забезпечує потужні можливості налаштування.

Бібліотеки: Linux використовує різноманітні бібліотеки, що містять функції та процедури, які можуть бути використані програмами. Ці бібліотеки розширюють функціональність операційної системи та допомагають розробникам створювати програми більш ефективно.

Система контролю версій: Linux дозволяє користувачам використовувати різноманітні системи контролю версій, такі як Git, для керування та відстеження змін в програмах та інших файлових ресурсах.

Мережеві інструменти: Операційна система Linux має вбудовані інструменти для налаштування мережі, моніторингу трафіку, налагодження мережевих підключень та багато іншого.

Віртуалізація: Linux підтримує різні технології віртуалізації, такі як KVM (Kernel-based Virtual Machine), Docker, а також віртуалізацію на рівні операційної системи з використанням контейнерів.

Безпека: Linux має високий рівень безпеки завдяки активному співробітництву спільноти та регулярним оновленням. Також, користувачі можуть налаштовувати права доступу та обмеження для забезпечення безпеки своєї системи.

Ці різноманітні складові роблять операційну систему Linux потужною, гнучкою та адаптованою до різноманітних потреб користувачів. Відкритий вихідний код дозволяє глобальній спільноті співпрацювати над розвитком та вдосконаленням системи, роблячи її більш інноваційною і сучасною.

Багато людей задають питання про необхідність використання Linux, враховуючи те, що більшість комп'ютерів, ноутбуків і серверів постачаються з іншими операційними системами, які працюють задовільно.

Щоб дати відповідь на це питання, важливо звернутися до переваг, які пропонує Linux порівняно з іншими операційними системами. Припустимо, ви зіткнулися з такими проблемами, як віруси, шкідливе програмне забезпечення, низька продуктивність, системні збої або високі витрати на обслуговування та ліцензії.

Linux став однією з найнадійніших комп'ютерних екосистем на планеті, і його популярність зростає. Одним з основних переваг Linux є його нульова вартість, що дозволяє встановлювати цю операційну систему на стільки комп'ютерів, скільки потрібно, без витрат на програмне забезпечення або ліцензійні витрати для серверів.

За допомогою Linux, ви можете отримати стабільну та безпечну платформу, яка не піддається впливу вірусів та шкідливого ПЗ. Linux також пропонує широкий вибір дистрибутивів, що задовольнять різні потреби користувачів.

Крім того, Linux дозволяє користувачам збільшити продуктивність, використовуючи різні оптимізаційні та настройки, а також використовувати його для великої кількості завдань, від серверних платформ до інтернет-пристроїв та вбудованих систем.

Загалом, Linux пропонує надійне, безкоштовне і потужне рішення для користувачів, які шукають альтернативу іншим операційним системам. Це можливо допоможе покращити продуктивність, забезпечити безпеку та знизити загальні витрати на програмне забезпечення та обслуговування.

Давайте розглянемо вартість сервера з операційною системою Linux порівняно з Windows Server 2016. Ціна ліцензії на Windows Server 2016 Standard

становить приблизно 900 доларів США, і це лише вартість самої операційної системи. До цього необхідно додатково придбати ліцензії користувачів (CAL) і інше програмне забезпечення, таке як бази даних, веб-сервери, поштові сервери та інше. Наприклад, CAL для одного користувача Windows Server 2016 коштує близько 40 доларів США. Якщо вам потрібно ліцензувати 10 користувачів, то це ще 400 доларів додаткових витрат.

За використання сервера з операційною системою Linux, всі ці витрати відпадають, оскільки Linux є безкоштовним та має відкритий вихідний код. Установка повноцінного веб-сервера (включаючи сервер бази даних) на Linux займає всього кілька клацань або команд.

Крім нульової вартості, Linux набагато менш вразливий до атак і вірусів, порівняно з Windows Server. Це забезпечує більшу стабільність та надійність роботи. Зазвичай, перезавантаження сервера Linux потрібні лише під час оновлення ядра, і сервер може працювати роками без перезавантаження.

Застосування регулярних оновлень дозволяє забезпечити стабільну та безпечну роботу сервера з Linux. Всі ці переваги роблять Linux привабливим вибором для тих, хто шукає економічно ефективну та надійну операційну систему для своїх серверів.

2.2 Основні відомості та характеристики

Операційна система UNIX має два основних об'єкти, з якими працює користувач - файли та процеси. Ці об'єкти взаємодіють між собою і визначають архітектуру операційної системи.

У світі UNIX всі дані зберігаються у файлах, і ця ідеологія відображається в принципі "Все є файлом". Кожен об'єкт, будь то текстовий документ, програмний код або пристрій, що підключений до системи, визначається файлом. Отримання доступу до пристроїв також здійснюється шляхом зчитування та запису спеціальних файлів.

Під час виконання програми операційна система зчитує виконуваний код з відповідного файлу та передає йому управління. Таким чином, функціональність операційної системи визначається виконанням відповідних процесів. Кожен

процес має свій власний контекст та пам'ять, що дозволяє йому працювати незалежно від інших процесів.

Під час звернення до файлів на диску, операційна система забезпечує наявність необхідного коду пам'яті для файлової підсистеми, що дозволяє користувачам взаємодіяти з файловою системою та зберігати дані безпечно та ефективно.

Таким чином, файлова і процесна підсистеми UNIX взаємодіють між собою, що дозволяє забезпечити ефективне та надійне функціонування операційної системи. Файли зберігають дані, а процеси забезпечують їх обробку та взаємодію з користувачем. Ця організація роботи з файлами та процесами додає UNIX гнучкості, надійності та продуктивності.

Дистрибутиви, що базуються на операційних системах Unix, володіють рядом характеристик, які роблять їх потужними та універсальними для різних завдань:

Величезна багатозадачність: Дистрибутиви Unix забезпечують підтримку великої кількості процесів, які працюють одночасно в ізольованих адресних просторах віртуальної пам'яті. Це дозволяє виконувати багато завдань одночасно та ефективно використовувати ресурси системи.

Підтримка багатьох користувачів: ОС Unix здатна обслуговувати одночасно багато користувачів, кожен з яких має власний окремий сеанс роботи. Це дозволяє віддалено або локально підключатися до системи та працювати з нею незалежно.

Асинхронні процеси: Дистрибутиви Unix підтримують асинхронне виконання процесів, що дозволяє ефективно працювати з завданнями, які можуть бути виконані паралельно та не залежать одне від одного.

Ієрархічна файлова система: Операційні системи Unix мають ієрархічну структуру файлової системи, що дозволяє організовувати файли та директорії у логічних групах. Це забезпечує легку організацію та керування даними.

Підтримка незалежних пристроїв вводу-виводу: В дистрибутивах Unix є спеціальні файли пристроїв, які дозволяють програмам взаємодіяти з пристроями вводу-виводу, такими як принтери, мережеві адаптери, диски тощо.

Стандартний інтерфейс: Unix надає стандартизований інтерфейс для програм та користувачів, що спрощує розробку та виконання програм на різних дистрибутивах.

Засоби обліку використання системи: Дистрибутиви Unix мають вбудовані засоби для відстеження використання ресурсів системи, що допомагає здійснювати моніторинг та налагодження.

Ці характеристики роблять дистрибутиви Unix привабливими для серверних рішень та роботи з багатозадачними завданнями, а також надають гнучкість та надійність для різних сфер діяльності.

Архітектура ОС Unix

Архітектура операційної системи Unix є багаторівневою, що дозволяє розділити функції та завдання на різні рівні для більшої ефективності та легшого керування. На нижньому рівні працює ядро ОС, яке безпосередньо взаємодіє з апаратною складовою комп'ютера та забезпечує управління ресурсами. Функції ядра доступні через інтерфейс системних викликів, що утворює другий рівень.

На третьому рівні розташовані командні інтерпритатори, команди та утиліти системного адміністрування, драйвери та протоколи, що відповідають за різноманітні завдання в операційній системі. Цей рівень утворює розширений інтерфейс між ядром і прикладним програмним забезпеченням.

Останній, зовнішній рівень складають прикладні програми користувача, мережеві та системні служби, а також утиліти, які виконують певні функції. Цей рівень взаємодіє безпосередньо з користувачем та забезпечує доступ до різноманітних додатків та сервісів.

Така багаторівнева архітектура дозволяє забезпечити чітке розділення обов'язків та створення ієрархії функцій в операційній системі. Кожен рівень виконує свої завдання, а зв'язки між ними дозволяють оптимально керувати ресурсами та забезпечувати стабільну та ефективну роботу операційної системи Unix.

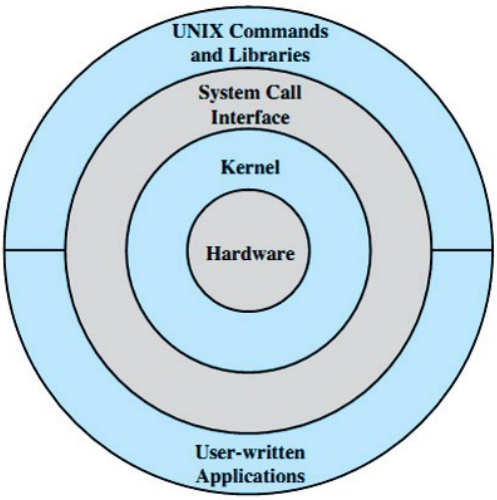

Рисунок 2.1 - Схематичне зображення рівнів ядра

Основні функції ядра

Основні функції ядра операційної системи UNIX, яке може бути реалізоване як монолітне або модульне, охоплюють важливі аспекти керування та координації роботи всієї системи. Для забезпечення ефективної та надійної роботи операційної системи, ядро виконує наступні основні завдання:

Планування та перемикання процесів: Ядро відповідає за вибір та призначення процесів для виконання на процесорі. Воно керує часом, який кожен процес отримує для виконання, а також переключенням між процесами для забезпечення мультизадачності.

Керування пам'яттю: Ядро відповідає за виділення та управління пам'яттю, необхідною для виконання процесів. Воно забезпечує ізоляцію процесів та контроль за їх доступом до пам'яті.

Обробка переривань: Ядро взаємодіє з апаратурою та драйверами, щоб обробити переривання, які виникають з зовнішніх пристроїв чи подій. Це дозволяє системі реагувати на зміни та події в реальному часі.

Низькорівнева підтримка пристроїв (через драйвери): Ядро включає драйвери, які забезпечують низькорівневу комунікацію та управління пристроями, такими як диски, мережеві адаптери, принтери тощо.

Управління дисками: Ядро контролює доступ до дискової пам'яті та забезпечує організацію файлових систем для зберігання даних.

Буферизація даних: Ядро забезпечує механізм буферизації, що дозволяє ефективно обробляти та передавати дані між пристроями та пам'яттю.

Ініціалізація системи: Ядро відповідає за запуск та ініціалізацію всіх компонентів системи під час старту.

Синхронізація процесів та забезпечення засобів міжпроцесної взаємодії: Ядро вирішує питання взаємодії між процесами, синхронізації доступу до ресурсів та розв'язання конфліктів.

Окрім основних функцій, ядро операційної системи UNIX надає доступ до деяких спеціальних функцій для процесів користувача шляхом системних викликів. Ці системні виклики дозволяють виконувати завдання, які користувальницький процес не може виконати самостійно або які потребують привілеїв доступу.

Два дуже важливі системні виклики - fork() і exec() - відіграють ключову роль у процесі запуску нових процесів.

fork(): Коли процес здійснює виклик fork(), ядро створює практично ідентичну копію цього процесу. Таким чином, отримується новий процес, відомий як "дочірній" процес, який має ті ж властивості та стан, що й батьківський процес. Після виклику fork(), обидва процеси - батьківський та дочірній - продовжують виконуватися, але вони мають різний процесний ідентифікатор (PID).

exec(): Коли процес здійснює виклик exec(program), ядро запускає програму program, яка замінює поточний процес. Іншими словами, виконання поточного процесу завершується, і його заміняє виконання програми program. Це дозволяє запускати нові програми та забезпечує зміну контексту виконання.

Зазвичай процеси користувача у системі Linux створюються шляхом виклику fork(), а потім виклику exec(), щоб запустити нову програму, а не просто копію існуючого процесу. Наприклад, коли ви виконуєте команду ls у вікні терміналу, оболонка спочатку здійснює виклик fork(), щоб створити копію себе, а потім нова

копія оболонки здійснює виклик exec(ls), щоб запустити команду ls. Така послідовність дозволяє виконувати команди та програми у системі.

На рис. 2.2 показано послідовність процесів та системних викликів для запуску таких програм, як ls.

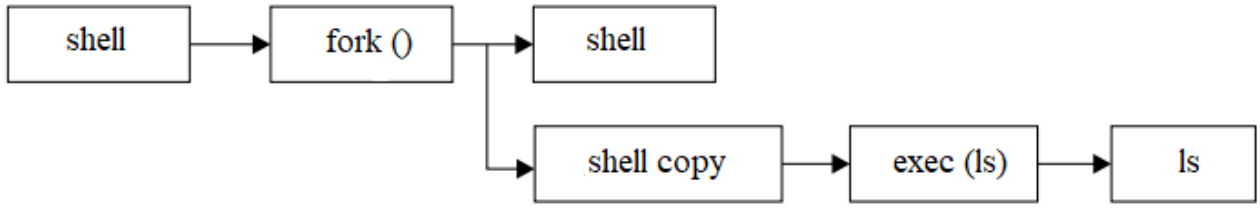

Рисунок 2.2 – Схематичне зображення запуску нового процесу

Поміж основних функцій ядра операційної системи UNIX, також існує підтримка процесів користувача, зокрема псевдовлаштувань, які відрізняються від звичайних системних викликів.

Псевдовлаштування - це механізм, який надає користувачам можливість взаємодіяти з деякими функціями ядра через спеціальні файли, які виглядають і працюють як звичайні пристрої. Вони є результатом програмної реалізації в ядрі та дозволяють виконувати складні операції, які потребують привілеїв або безпеки, в контексті користувача.

Прикладом псевдовлаштувань можуть бути файли в /dev, такі як /dev/random, який генерує випадкові числа. Реалізація цього пристрою в процесі користувача була б складною з точки зору безпеки та відповідності стандартам, тому його реалізують в ядрі для забезпечення високого рівня надійності.

Ще одна важлива функція ядра - розподіл та управління оперативною пам'яттю. Ядро відповідає за розподіл доступної пам'яті між різними процесами і забезпечує, щоб кожен процес мав доступ лише до своєї виділеної області пам'яті. Це забезпечує відокремлення процесів і запобігає взаємовпливу між ними.

Таким чином, псевдовлаштування та управління пам'яттю є важливими аспектами функціональності ядра UNIX, які допомагають забезпечити безпеку, надійність та відокремлення процесів користувача в операційній системі.

Ядро операційної системи відповідає за керування різними аспектами системи, і його функції можна розділити на чотири основні галузі:

Управління процесами: Ядро визначає, яким процесам та у якому порядку дозволяється використовувати центральний процесор. Воно планує та перемикає процеси, дозволяючи їм виконуватися в ізольованих часових інтервалах, що забезпечує багатозадачність та одночасну роботу багатьох процесів.

Управління пам'яттю: Ядро відстежує стан всієї пам'яті, визначаючи, яка частина зайнята процесами, яка виділена для спільного використання та яка вільна. Воно забезпечує аллокацію та деаллокацію пам'яті для процесів та контролює доступ до неї.

Керування драйверами пристроїв: Ядро служить інтерфейсом між апаратними засобами, такими як жорсткий диск, мережева карта, та процесами. Воно забезпечує взаємодію з пристроями та контролює їх функціонування, зокрема виконання операцій вводу-виводу.

Системні виклики та підтримка: Процеси взаємодіють з ядром, викликаючи системні виклики. Ці виклики дозволяють процесам отримати доступ до служб ядра, наприклад, створення нового процесу, керування файлами, мережевими з'єднаннями та іншими операціями, які вимагають привілеїв.

Загалом, ядро операційної системи відіграє важливу роль у керуванні ресурсами та забезпеченні взаємодії між різними компонентами системи, що дозволяє ефективно та надійно працювати з багатозадачними завданнями та взаємодіяти з апаратними засобами.

Тепер коротко розглянемо кожну з цих областей. Управління процесами означає зупинку, запуск, поновлення та припинення роботи процесів. Поняття, що стоять за процесами запуску та припинення процесів, дуже прості. Трохи складніше описати те, як процес використовує центральний процесор у нормальному режимі роботи. У всіх сучасних операційних системах кілька процесів функціонують «одночасно». Наприклад, в один і той же час ви можете запустити на комп'ютері браузер та відкрити файл у ворді. Але насправді все не так,

як виглядає: процеси, які відповідають за ці додатки, зазвичай не запускаються в точності в один момент часу.

Якщо розгладати систему з одним центральним процесором то його можуть використовувати кілька процесів, але в кожен певний момент часу тільки один процес може дійсно застосовувати процесор. В житті, на практиці, кожен процес використовує процесор протягом дуже малої долі секунди, а потім припиняється. Після цього інший процес застосовує процесор так само. Далі настає черга третього процесу і т.д. Дія, при якому будь-який процес передає іншому процесу управління процесором, називається перемиканням контексту. Кожен відрізок часу – квант часу, що надає процесу достатньо часу для виконання необхідних йому обчислень. Кванти часу дуже малі, людина їх, просто, не сприймає і здається, що в системі одночасно виконується кілька процесів, зазвичай, це називають «багатозадачність».

Ядро відповідає за перемикання контексту. Щоб зрозуміти, як це працює, уявімо ситуацію, в якій процес, який запущений в режимі користувача, але квант його часу вже закінчується.

Процес перемикання контексту в операційній системі включає кілька етапів, які забезпечують ефективне управління процесами і ресурсами комп'ютера:

В першому етапі процесор, спираючись на внутрішній таймер, перериває виконання поточного процесу та переходить у режим ядра операційної системи. Поточний стан процесора і пам'яті записуються ядром, щоб забезпечити відновлення перерваного процесу пізніше.

Далі, ядро виконує необхідні завдання, які можуть виникнути протягом попереднього кванта часу, такі як обмін даними або операції введення-виведення.

Після цього, ядро вибирає зі списку готових до запуску процесів один із них, який готовий продовжити своє виконання.

Ядро підготовлює пам'ять для нового процесу та готує процесор для його виконання.

Далі, ядро повідомляє процесору, скільки часу потрібно виділити новому процесу на його виконання (квант часу).

Після цього, ядро знову переводить процесор у режим користувача та передає управління новому процесу, що дозволяє йому продовжити своє виконання.

Цей процес перемикання контексту відбувається дуже швидко, що дозволяє багатозадачній системі забезпечувати враження одночасного виконання кількох процесів користувачем. Ядро операційної системи відповідає за ефективне розподілення ресурсів і керування процесами, забезпечуючи оптимальну продуктивність системи.

Перемикання контексту грає важливу роль у визначенні активності ядра операційної системи. Ядро працює в періоди між квантами часу, що виділяються для виконання процесів. Коли наставає момент перемикання контексту, ядро переходить у дію для збереження поточного стану процесу та підготовки наступного процесу для виконання. Цей процес забезпечує ефективне керування ресурсами і дозволяє системі працювати з багатьма процесами практично одночасно. Крім того, перемикання контексту відбувається настільки швидко, що користувач сприймає роботу системи як багатозадачну, незалежно від того, що фактично процеси виконуються послідовно в межах своїх квантів часу.

У системах з кількома процесорами управління ресурсами стає більш складним завданням для ядра операційної системи. Ядро повинно забезпечити правильне управління пам'яттю, дозволяючи кожному процесу користувача працювати з власною областю пам'яті, що не перетинається з областями інших процесів. Також, ядро повинно забезпечити спільне використання пам'яті між деякими процесами, якщо це необхідно, а також використання додаткового дискового простору для збереження недоступної в пам'яті інформації.

Сучасні процесори мають модуль управління пам'яттю (MMU), що дозволяє використовувати віртуальну пам'ять або swap. Завдяки цьому процесам дозволяється звертатися до пам'яті, як якби вони мали в своєму розпорядженні всю машину, хоча фізично пам'ять може бути розподілена між декількома процесами.

Ядро відповідає за ініціалізацію, постійне підтримання та зміну карт пам'яті, які перекладають віртуальні адреси процесів в фізичні адреси в пам'яті комп'ютера. При перемиканні контексту, ядро має оновити ці карту для наступного процесу, що

готується до виконання, забезпечуючи коректну роботу процесів та ефективне використання ресурсів системи.

Системні виклики.

Системні виклики відіграють важливу роль у забезпеченні взаємодії процесів користувача з ядром операційної системи. Вони надають програмам користувача можливість звертатися до захищених процедур ядра для виконання системних функцій.

Ці виклики забезпечують різноманітні операції, такі як зіставлення дій користувача з запитами драйверів пристроїв, створення та припинення процесів, операції введення-виводу, доступ до файлів та дисків, а також підтримку функцій терміналу.

При виклику системного виклику, процес користувача перетворюється на захищений процес в режимі ядра, що дозволяє виконувати захищені процедури ядра для виконання запитів. Це забезпечує безпеку та контроль доступу до ресурсів системи.

Також важливо зазначити, що доступ до багатьох системних викликів можливий через командний інтерпретатор (наприклад, bash або shell), що спрощує взаємодію користувачів з операційною системою і дозволяє здійснювати різні дії та операції.

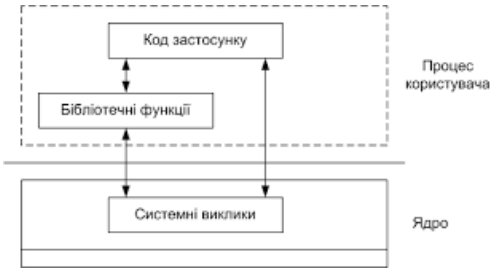

Рисунок 2.3 – Схематичне відображення взаємодії викликів

Користувальницькі процеси та процеси ядра

Користувацькі процеси можна розділити на два рівні, що забезпечують взаємодію з ядром операційної системи:

Рівень захищеності: Користувацькі процеси працюють в захищеному середовищі, що дозволяє їм ізоляцію від інших процесів користувача. Вони не

мають прямого доступу до процедур ядра та пам'яті ядра. Замість цього, користувацькі процеси можуть звертатися до функцій ядра через системні виклики, які дозволяють виконання захищених операцій та доступ до системних ресурсів.

Простір ядра: Це особлива область пам'яті, в якій процеси ядра реалізують служби ядра. Процеси, які працюють в просторі ядра, вважаються працюючими у режимі ядра. Цей простір є привілейованим, і користувачі не мають прямого доступу до нього. Взаємодія користувача з простором ядра відбувається лише через системні виклики, що дозволяють обмежений доступ до операцій та функцій ядра.

Користувальницький процес може перейти в режим ядра, коли здійснює системний виклик, що активує виконання коду ядра для виконання певної системної функції. Таким чином, користувач може взаємодіяти з операційною системою і виконувати спеціальні операції, які потребують привілейованого доступу, через обмежений інтерфейс системних викликів.

Обмін даними між простором ядра та користувальницьким простором.

Для передачі даних між процесами користувача та ядром, які не мають загального адресного простору пам'яті, використовується спеціальний механізм. При виконанні системного виклику, аргументи виклику та ідентифікатор відповідної процедури ядра передаються з простору користувача в простір ядра.

Ідентифікатор процедури ядра зазвичай передається через апаратний регістр процесора або через стек. Це дозволяє ядрі знати, яка конкретна операція повинна бути виконана.

Аргументи системного виклику передаються через спеціально виділену область пам'яті, яку називають "область виклику процесу". Ця область служить для обміну даними між процесом користувача та ядром. Користувач передає необхідні дані у цю область перед здійсненням системного виклику, а ядро зчитує ці дані під час виконання відповідної операції.

Таким чином, механізм передачі даних між процесами користувача та ядром забезпечує безпечний і контрольований обмін інформацією між цими двома рівнями операційної системи.

Користувацька область процесу містить різноманітну інформацію, яка відображає стан та параметри самого процесу. Ось деякі з цих даних:

Кореневий та поточний каталоги: Вказівник на кореневий каталог процесу, який визначає базовий каталог, від якого починається шлях до інших файлів та каталогів. Поточний каталог вказує на каталог, в межах якого процес виконує свої операції відносно файлів.

Аргументи поточного системного виклику: Дані та параметри, передані процесом під час виконання системного виклику. Ці аргументи містять необхідну інформацію для ядра, щоб виконати відповідну операцію.

Розміри сегмента тексту, даних та стеку: Визначають розміри сегментів пам'яті процесу, які містять виконуваний код програми (сегмент тексту), дані (сегмент даних) та стек викликів функцій.

Покажчик на запис у таблиці процесів: Вказівник на запис процесу у таблиці процесів, яка містить інформацію для планувальника процесів, наприклад, його пріоритет та інші важливі параметри.

Таблиця дескрипторів файлів користувача: Містить інформацію про відкриті файли процесу, такі як дескриптори файлів, що дозволяють процесу звертатися до них для зчитування та запису даних.

Стек ядра для процесу: Цей стек використовується, коли процес виконує системний виклик, і потрібна додаткова пам'ять для передачі параметрів та збереження контексту.

Ці дані допомагають операційній системі відстежувати, контролювати та керувати процесами користувача, забезпечуючи ефективну та безпечну роботу системи.

Процесу користувача заборонений безпосередній доступ до простору ядра, але ядро має повний контроль над простором процесу користувача. Простір користувача представляє область оперативної пам'яті, яку ядро виділяє для збереження даних та коду процесів користувача. Завдяки цьому, процеси користувача можуть працювати із своїми власними даними та інструкціями, і ці дані залишаються недоступними для інших процесів користувача.

У системі Linux, більшість операцій та дій відбуваються у просторі користувача. Користувачі взаємодіють з системою шляхом запуску процесів, виконання програм та взаємодії з інтерфейсами. Ці процеси користувача можуть мати різні завдання та функціональність залежно від потреб користувачів.

Щоб забезпечити цю ізоляцію та безпеку, системні компоненти, які представляють процеси користувача, організовані у вигляді сервісного рівня або шару. Цей шар відповідає за керування процесами користувача, управління пам'яттю, контроль доступу та інші операції, що забезпечують незалежність та ефективну роботу процесів. Така архітектура дозволяє кожному процесу користувача працювати відокремлено, зберігаючи при цьому безпеку та стабільність всієї системи.

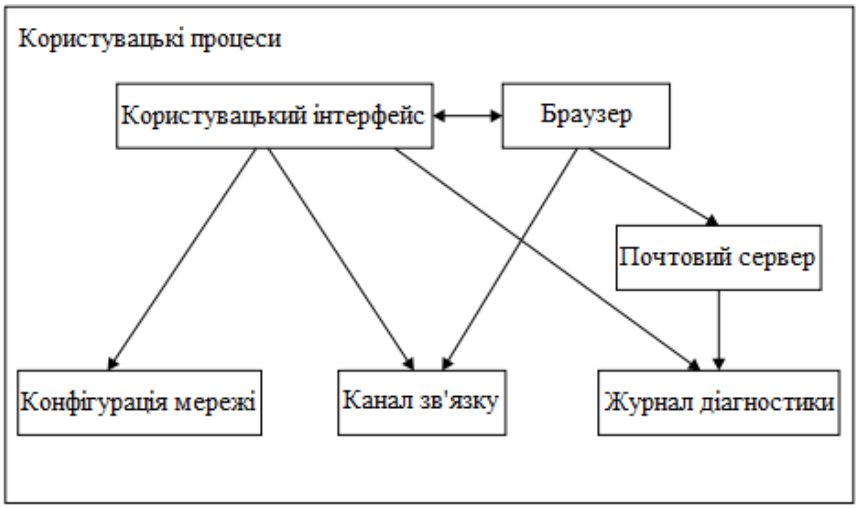

Рисунок 2.4 - Типи процесів та взаємодій

На рис. 2.4 показаний зразковий набір компонентів, пов'язаних між собою та взаємодіючих із системою Linux. Рисунок 7 є вкрай спрощеною схемою, оскільки показані лише шість компонентів, але ви можете помітити, що верхні компоненти знаходяться найближче до користувача (інтерфейс користувача і браузер).

Архітектура системи може бути уявлена у вигляді трьох рівнів: нижнього, середнього та верхнього. Кожен рівень виконує свої функції та має взаємодію з іншими рівнями для забезпечення роботи системи.

На нижньому рівні розміщені прості служби та компоненти, що виконують прості завдання. Сюди входять невеликі компоненти, що взаємодіють з апаратними засобами та забезпечують базові операції.

Середній рівень містить більші компоненти, такі як поштова служба та база даних, що виконують складніші завдання. Ці компоненти забезпечують взаємодію між верхнім та нижнім рівнями, а також здійснюють обробку та збереження даних.

На верхньому рівні знаходяться компоненти, що виконують складні завдання та контролюються безпосередньо користувачами. Це можуть бути програми інтерфейсу користувача та браузер, які надають доступ до функцій системи для користувачів.

Забезпечення взаємодії між компонентами відбувається за допомогою системних викликів та інтерфейсів. Компоненти звертаються один до одного, якщо це необхідно, знаходячись на тому ж або нижчому рівні. Така архітектура дозволяє системі працювати організовано та ефективно, розділяючи завдання між різними рівнями. При цьому компоненти можуть вести діагностичний журнал, що допомагає виявити проблеми та спростити відлагодження системи.

Структура файлової системи

Файлова система в операційних системах Linux та Windows має ієрархічну структуру каталогів та файлів, але між ними є кілька кардинальних відмінностей. В ОС Windows, кожен жорсткий диск або розділ на ньому має свою латинську літеру, і кожен з дисків представляє кореневий каталог з власною деревовидною ієрархією папок. При підключенні нового пристрою, такого як накопичувач, з'являється новий кореневий каталог з новою літерою та своєю структурою.

У ОС Linux зовсім інший підхід. Файлова система представлена єдиним кореневим каталогом, позначеним як слеш (/). Відповідно, у цій структурі диски не мають своїх окремих каталогів, а навпаки, каталоги виступають як "точки монтування" для дисків або накопичувачів. Коли підключається змінний носій або диск, його файл пристрою стає видимим у каталозі /dev/ (devices). Але щоб отримати доступ до вмісту цього пристрою, його спочатку потрібно змонтувати (підключити) у відповідну окрему директорію, зазвичай під іменем /mnt/ (mount).

Цей підхід дозволяє Linux більш гнучко керувати файловими ресурсами та забезпечує один об'єднаний ієрархічний кореневий каталог для всіх пристроїв та накопичувачів.

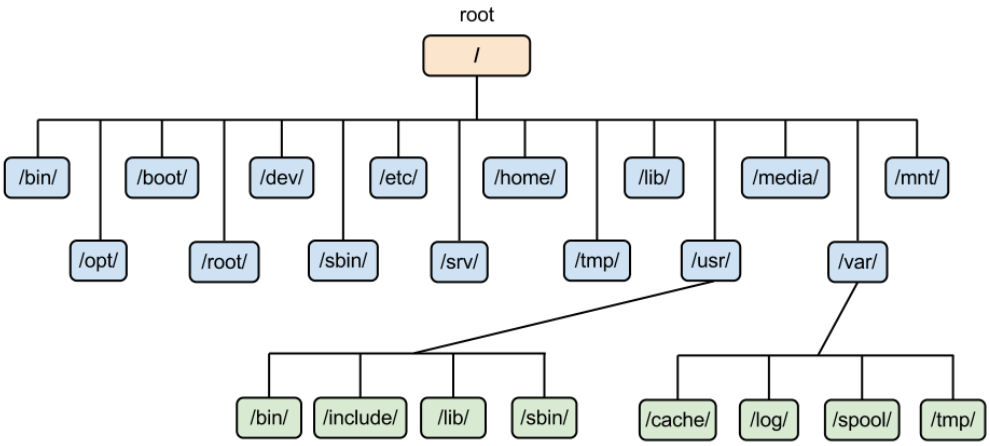

Рис.2.5 - Ієрархія файлової системи ОС Linux

На рис. 2.5 представлений спрощений варіант ієрархії каталогів, в якому показано декілька каталогів всередині /, /usr, /var. Варто звернути увагу, що всередині каталогу /usr, є такі самі назви, як у каталозі /.

Найбільш важливі підкаталоги в кореневому каталозі:

– /bin: є однією з основних та невід'ємних складових файлової системи у багатьох операційних системах Unix і подібних до них. Ця директорія відіграє ключову роль у функціонуванні операційної системи, оскільки саме тут розташовані виконуючі файли, які містять в собі виконуваний код різноманітних команд. Однією з найважливіших функцій директорії /bin є забезпечення доступу до основних системних команд, необхідних для взаємодії з операційною системою. Кожен виконуючий файл у цій директорії представляє собою готовий до запуску виконавчий код, який виконує конкретну команду. Серед них можна виділити такі ключові команди, як ls (виведення змісту поточного каталогу), cp (копіювання файлів), mv (переміщення або перейменування файлів), rm (видалення файлів або каталогів) та інші. Окрім базових системних команд, у директорії /bin також можна знайти виконавчі файли для взаємодії з файловою системою, мережевими протоколами, апаратним обладнанням та іншими ключовими аспектами

операційної системи. Наприклад, команди, які дозволяють взаємодіяти з пристроями вводу-виводу, такі як stdin та stdout, також можуть бути знайдені у цій директорії. Важливо відзначити, що директорія /bin є однією з перших, які система шукає під час виконання команд користувача. Це означає, що шлях до /bin часто включений у змінну середовища PATH, що полегшує виклик системних команд з будь-якої точки файлової системи.

– /etc: ще одна з найважливіших складових файлової системи в операційних системах Unix та подібних до них. Ця директорія відіграє критичну роль у забезпеченні стабільності та належної роботи системи, оскільки саме тут розміщуються файли конфігурації, які визначають параметри різних аспектів операційної системи. Основна функція директорії /etc - це зберігання конфігураційних файлів, які визначають поведінку різних компонентів операційної системи. Наприклад, у цій директорії можна знайти файли, які відповідають за налаштування мережі, ідентифікацію та авторизацію користувачів, конфігурацію служб та демонів, а також інші параметри системи. Файли конфігурації, розташовані у /etc, є важливим інструментом для керування та налагодження операційної системи з точки зору адміністраторів систем. Один із ключових аспектів /etc - це налаштування мережі. Файли, такі як /etc/network/interfaces чи /etc/sysconfig/network-scripts/ifcfg-eth0, містять параметри, які визначають IP-адреси, шлюзи та інші мережеві налаштування. Це важливо для забезпечення правильної комунікації між комп'ютерами та ресурсами в мережі. Крім того, директорія /etc також містить файли, які визначають параметри служб і демонів. Наприклад, файли /etc/apache2/apache2.conf чи /etc/nginx/nginx.conf містять налаштування для веб-серверів Apache і Nginx відповідно. Це дозволяє адміністраторам змінювати поведінку сервера залежно від їхніх потреб та вимог проекту. Крім цього, директорія /etc є місцем зберігання конфігураційних файлів для різноманітних програм та пакетів, встановлених на системі. Наприклад, /etc/apt/sources.list містить налаштування для системи управління пакетами APT в Debian-based дистрибутивах. Оскільки файли в /etc мають важливе значення для стабільності та ефективності операційної системи, зазвичай доступ до цієї

директорії має обмежений правами доступу. Це робить неможливим випадкові чи несанкціоновані зміни, забезпечуючи надійність системи та захист від можливих помилок чи вторгнень.

– /home: Директорія /home в системах Unix та подібних є не тільки простим резервуаром для особистих каталогів користувачів, але й гравцем з ключовою роллю в забезпеченні особистого простору для кожного із них. Ця директорія є необхідною складовою для організації та управління даними користувачів, і вона втілює стандартну структуру каталогів, що полегшує організацію і пошук інформації. Основна функція /home полягає у створенні та управлінні особистими каталогами для кожного користувача, який має доступ до системи. Кожен користувач отримує власну піддиректорію в /home, названу його ім'ям користувача (наприклад, /home/john або /home/mary). Це сприяє легкості ідентифікації та управління ресурсами для кожного користувача, а також дозволяє кожному користувачеві зберігати свої особисті файли, документи, налаштування та інші дані в окремому просторі. Стандартна структура каталогів в /home включає піддиректорії, такі як Documents, Downloads, Pictures, Music, і т.д., які допомагають користувачам логічно організувати свої дані. Це не тільки сприяє легшому доступу до різних категорій інформації, але і робить резервні копії та переміщення даних більш структурованими та зручними. Крім того, директорія /home може містити спільні ресурси або файли, доступні для спільного використання користувачами. Наприклад, у піддиректорії /home/shared може бути створено місце для обміну файлами між різними користувачами, що сприяє співпраці та обміну ресурсами. Забезпечення безпеки важливий аспект управління /home. Каталоги користувачів повинні мати обмежені права доступу, щоб уникнути несанкціонованого доступу до особистих даних. Зазвичай, кожен користувач має права на читання та запис лише до свого власного каталогу, що забезпечує приватність та безпеку даних. Окрім цього, директорія /home служить важливим компонентом у процесі роботи з обліковими записами користувачів, адмініструванням прав доступу та налаштуванням середовища для кожного користувача. Вона також відіграє ключову роль у реалізації політик безпеки та забезпеченні конфіденційності

інформації.

– /lib: Директорія /lib у системах Unix відіграє важливу роль у забезпеченні ефективності та стабільності операційної системи. Цей каталог містить бібліотеки, які представляють собою збірники програмного коду, призначені для використання виконуваними файлами різних програм. Бібліотеки в /lib використовуються для оптимізації роботи програм, а також для забезпечення їхньої сумісності та ефективного використання ресурсів операційної системи. Однією з ключових функцій /lib є забезпечення централізованого місця для зберігання та управління бібліотеками, які використовуються різними програмами. Це дозволяє уникнути дублювання коду та забезпечити взаємодію різних додатків зі спільними бібліотеками. Зокрема, багато виконуваних файлів може посилатися на одну і ту ж бібліотеку, розташовану в /lib, що сприяє економії місця та оптимізації ресурсів. Окрім того, /lib містить необхідні бібліотеки для завдань, таких як завантаження операційної системи або виконання базових функцій. Наприклад, тут можуть бути розташовані бібліотеки для роботи з файловою системою, мережею, криптографією та іншими системними ресурсами. Це дозволяє забезпечити необхідність в базових функціях під час роботи системи або виконання деяких операцій. Однією з ключових особливостей /lib є його роль у завданні динамічної лінкування. У багатьох системах, що використовують Unix-подібний підхід, бібліотеки в /lib можуть бути використані динамічно, коли програма виконується. Це означає, що виконуваний файл може взаємодіяти з бібліотеками лише тоді, коли це дійсно необхідно, що дозволяє економити пам'ять та збільшувати швидкість виконання. Окрім того, директорія /lib також може містити важливі файли для завантаження операційної системи, такі як ядро (ядро операційної системи) або інші компоненти, необхідні для початкової ініціалізації системи під час завантаження.

– /proc: Директорія /proc в операційних системах Unix є унікальною і важливою складовою файлової системи, оскільки вона дозволяє отримувати доступ до системної інформації та статистики у формі інтерфейсу "каталог-файл". Цей механізм дозволяє користувачам та системним адміністраторам отримувати

детальну інформацію про роботу ядра операційної системи, запущені процеси, а також інші параметри системи. Однією з ключових функцій /proc є надання доступу до інформації про запущені процеси в системі. В директорії /proc можна знайти підкаталоги з числовими назвами, які відповідають ідентифікаторам процесів (PID). Кожен такий каталог містить різні файли та підкаталоги, які містять інформацію про конкретний процес, таку як статус, власник, використання ресурсів, інформація про пам'ять та інші параметри. Це дозволяє користувачам в режимі реального часу відслідковувати та аналізувати активність процесів у системі. Крім того, /proc надає важливі дані про параметри ядра операційної системи. У цій директорії можна знайти файли, які надають інформацію про конфігурацію ядра, параметри планування завдань, використання пам'яті, інтервали таймерів та багато іншого. Наприклад, /proc/sys/kernel/sched_latency містить інформацію про затримку планування завдань у ядрі. Однією з унікальних можливостей /proc є можливість зчитування та зміни деяких параметрів ядра в реальному часі. Користувачі та адміністратори можуть використовувати цей механізм для оптимізації роботи системи або налаштування певних параметрів без перезавантаження системи. Директорія /proc також використовується для отримання інформації про системні ресурси, такі як пам'ять, процесор та мережа. Файли в цій директорії, такі як /proc/meminfo або /proc/net, надають детальну інформацію про використання ресурсів та стан системи. Окрім цього, /proc дозволяє взаємодіяти з різними підсистемами ядра. Наприклад, /proc/sys/fs/file-max містить обмеження на кількість відкритих файлів в системі, а /proc/sys/net/ipv4/ip_forward вказує, чи включений функціонал пересилання IP-пакетів. Необхідно також відзначити, що /proc є віртуальною файловою системою, тобто вміст цієї директорії створюється динамічно під час роботи системи і відображає поточний стан ядра та процесів.

– /sys: Подібний до /proc, надає інтерфейс для пристроїв та системи.

– /sbin: Містить системні файли, що виконуються, які відповідають за управління системою.

– /tmp: Використовується як сховище для тимчасових файлів, доступних

для всіх користувачів.

– /usr: Директорія /usr є однією з найважливіших і обширних частин файлової системи в операційних системах Unix, зокрема, у системі Linux. Ця директорія виконує ключову роль у зберіганні та організації різноманітних компонентів операційної системи, а також додаткового програмного забезпечення, розробленого спільнотою або встановленого користувачем. Однією з ключових функцій /usr є зберігання основної частини операційної системи, включаючи бібліотеки, заголовочні файли, та інші ресурси, які використовуються для компіляції та виконання програм. Наприклад, /usr/include може містити заголовочні файли, необхідні для розробки програмного забезпечення, тоді як /usr/lib може містити бібліотеки, які використовуються під час компіляції програм. Окрім того, /usr містить ієрархію каталогів, яка включає різні підкаталоги для додаткового програмного забезпечення. Наприклад, /usr/bin містить виконувані файли для додаткових програм, /usr/sbin - виконувані файли для системних служб та утиліт, а /usr/share містить ресурси, такі як зображення, звуки, локалізаційні файли, тощо. Ця структура організації сприяє порядку та легкості управління додатковими програмами та ресурсами системи. Ще однією важливою складовою /usr є /usr/local, який використовується для зберігання локально встановлених програм та додаткового програмного забезпечення, що відмінюється від стандартного системного програмного забезпечення. Це дозволяє користувачам та адміністраторам легко встановлювати та управляти локальними версіями програм без втручання в системні компоненти. Директорія /usr також є місцем для розташування додаткових репозиторіїв та джерел для систем управління пакетами. Системи Linux, такі як apt (використовується в Debian та Ubuntu) або yum (використовується в CentOS та Fedora), можуть звертатися до /usr для отримання інформації про пакети та їх залежності, а також для зберігання встановлених пакетів. Ще однією важливою функцією /usr є надання доступу до документації для встановлених програм та бібліотек. Зазвичай, /usr/share/doc містить підкаталоги для кожного пакету, які містять різні документи, README-файли, приклади коду та інші матеріали. Важливо відзначити, що /usr може бути монтуваний з різних дисків

чи файлових систем, що робить його підходящим для розділення простору для програмного забезпечення та даних.

– /var: Цей підкаталог змінюється, і він використовується для зберігання інформації під час роботи системи, такої як журнали, відстеження активності користувачів і т. д.

– /boot: Містить файли завантажувача ядра, які стосуються першої стадії запуску системи Linux.

– /media: Основна точка підключення для знімних пристроїв, таких як флешнакопичувачі.

– /opt: Може містити додаткове програмне забезпечення сторонніх розробників.

– /usr/include: Містить головні файли, які використовуються компілятором C.

Для аналізу мною була обрана ОС Debian, тому що вона являється найпоширенішою в серверному сегменті та пакети Wireguard там знаходяться в стандартному репозиторії, а система управління конфігураціями розроблялась саме для серверного сегменту і я вважаю, що в десктопному середовищі з Ansible робити нічого.

3 АНАЛІЗ ПРОТОКОЛУ WIREGUARD

3.1 Віртуальні приватні мережі та протокол Wireguard

Віртуальна приватна мережа (VPN) - це технологія, яка дозволяє створювати захищений канал зв'язку між двома або більше пристроями через Інтернет. VPN-з'єднання зазвичай шифрується, що робить його недоступним для сторонніх очей.

VPN-мережі використовуються для різних цілей, включаючи:

– Безпека та конфіденційність: VPN-з'єднання може допомогти захистити ваші дані від перехоплення хакерами або іншими зловмисниками;

– Доступ до заблокованого контенту: VPN-з'єднання можна використовувати для доступу до веб-сайтів і сервісів, які заблоковані в вашій країні;

– Географічний перехоплення: VPN-з'єднання можна використовувати для зміни вашої IP-адреси, що може бути корисно для перегляду контенту, доступного лише в певній географічній зоні.

VPN-з'єднання безпечні лише настільки, наскільки безпечне саме програмне забезпечення. Фахівці з кібербезпеки традиційно докладають критику до програм для VPN. Однією з причин цього є те, що більшість VPN-програм є надзвичайно складними. Чим складче програмне забезпечення, тим важче провести перевірку на предмет можливих проблем з безпекою.

Існує два основних типи VPN-мереж:

Тунель-базовані VPN: Ці мережі використовують тунель, щоб створити захищений канал між двома пристроями. Тунель шифрується, що робить його недоступним для сторонніх очей.

Мережеві VPN: Ці мережі створюють віртуальну мережу поверх існуючої мережі. Це дозволяє пристроям всередині віртуальної мережі спілкуватися один з одним, як ніби вони були фізично підключені до однієї мережі.

VPN-мережі можна використовувати на будь-якому пристрої, який має доступ до Інтернету. Найбільш поширеними платформами для VPN є комп'ютери, смартфони та планшети.

Робота VPN-протоколів ґрунтується на виконанні двох ключових функцій: авторизація (перевірка автентичності) і шифрування. Перша функція гарантує, що ваш пристрій встановлює зв'язок лише з надійним VPN-сервером, забезпечуючи високий рівень безпеки. Друга функція - шифрування даних, запобігає стороннім особам отримувати доступ до вашого трафіку збільшуючи конфіденційність.

Однак VPN-протоколи використовують різні стандарти шифрування та методи перевірки автентичності. Ця різниця впливає на швидкість та рівень безпеки з'єднань користувачів VPN. Кожен протокол також має свої власні правила обробки можливих помилок, що впливає на стабільність та надійність підключень.

Існує кілька розповсюджених VPN-протоколів, які використовуються для забезпечення безпеки та конфіденційності в Інтернеті. Ось деякі з них:

OpenVPN: OpenVPN є одним з найпопулярніших та найбільш універсальних VPN-протоколів. Він відкритий, безкоштовний та дуже безпечний. OpenVPN підтримує різні методи шифрування, включаючи AES і Blowfish. Він працює на багатьох операційних системах і може бути налаштований для різних рівнів безпеки.

IPsec (Internet Protocol Security): IPsec - це набір протоколів для створення безпечних тунелів для передачі IP-трафіку. Він часто використовується на рівні маршрутизаторів і мережевих пристроїв. IPsec може працювати в різних режимах, таких як тунельний та транспортний, і підтримує різні методи шифрування та аутентифікації.

L2TP/IPsec (Layer 2 Tunneling Protocol): L2TP є протоколом для створення тунелів, який часто використовується разом з IPsec для підвищення рівня безпеки. Він зазвичай використовується на пристроях з підтримкою IPsec.

PPTP (Point-to-Point Tunneling Protocol): PPTP є одним з найстаріших VPN-протоколів і використовується через вбудовану підтримку в багатьох операційних системах. Він має менший рівень безпеки в порівнянні з іншими протоколами і не рекомендується для важливих даних.

SSTP (Secure Socket Tunneling Protocol): SSTP є протоколом, розробленим Microsoft, і в основному використовується на платформах Windows. Він використовує SSL для шифрування і забезпечує високий рівень безпеки.

IKEv2 (Internet Key Exchange version 2): Цей протокол використовується для створення VPN-з'єднань і може бути використаний на різних платформах. Він підтримує різні методи шифрування і відомий своєю стійкістю до втрати з'єднання.

WireGuard: WireGuard - відносно новий VPN-протокол, який вже здобув значну популярність серед експертів у галузі, проект з відкритим кодом який доступний на GitHub. Цей протокол розроблено з огляду на швидкість, сучасність та безпеку, що робить його привабливим вибором для тих, хто шукає надійне рішення для VPN. Початково WireGuard був створений для ядра Linux, але зараз він доступний для різних платформ, включаючи Windows, macOS, BSD, iOS та Android, і активно використовується на них. Вперше він був випущений у 2016 році Джейсоном А. Доненфельдом і з того часу отримав широке визнання в галузі VPN.

Джейсон Доненфельд почав розробляти WireGuard в 2015 році, коли він вирішив створити новий VPN-протокол зі спрощеними та ефективними концепціями, що відрізнялися від традиційних рішень, таких як IPsec і OpenVPN.

Джейсон Доненфельд був незадоволений складністю та обсягом коду в традиційних рішеннях, таких як IPsec і OpenVPN. Він вважав, що це ускладнює аудит та розуміння безпеки протоколів. Також, він спостерігав низьку продуктивність та швидкодію в цих системах, що не задовольняло його очікувань щодо ефективності. Доненфельд шукав спрощені та ефективні рішення, які б мали мінімальний код та принципово новий підхід до побудови VPN-протоколу. Він бажав створити щось, що було б легким для розгортання та конфігурації, забезпечуючи при цьому високу безпеку та швидкодію.

Доненфельд обрав ядерний простір Linux як основну платформу для реалізації WireGuard. Це дозволило йому використовувати переваги уже наявної інфраструктури ядра та отримувати підтримку від спільноти Linux. WireGuard також був спроектований з великою увагою до безпеки та простоти налаштування,

що робить його привабливим для різних застосувань, від особистого використання до корпоративних мереж.

Проект WireGuard став публічним у 2016 році, коли Доненфельд опублікував його в якості ряду патчів до ядра Linux. З тих пір WireGuard отримав значну увагу та популярність в спільноті розробників і користувачів VPN. Він був інтегрований в основну гілку ядра Linux та підтримується на багатьох інших платформах, таких як Windows, macOS, Android та багатьох різних дистрибутивах Linux.

За декілька років з моменту створення WireGuard став дуже популярним рішенням для створення безпечних тунелів VPN, завдяки своїй ефективності та простоті використання.

На відміну від деяких застарілих та менш безпечних протоколів, WireGuard забезпечує високі швидкості та одночасно підвищує рівень безпеки. Цей протокол розроблений як універсальне рішення VPN, яке може бути використане як на вбудованих мережевих інтерфейсах, так і на потужних суперкомп'ютерах, роблячи його ідеальним вибором для різних сценаріїв. Його гнучкість також заслуговує на увагу, оскільки він здатен швидко та легко підключатися та перепідключатися, навіть під час переміщення між різними мережами.

WireGuard - це комунікаційний протокол, який створює безпечний зашифрований канал між двома або більше мережевими інтерфейсами. Він використовує найсучаснішу криптографію, включаючи Curve25519 для обміну ключами, ChaCha20 для шифрування та Poly1305 для перевірки автентичності повідомлень (MAC). WireGuard також був створений як простий та ефективний протокол з мінімальною базою коду та низьким навантаженням на процесор.

WireGuard має декілька важливих особливостей, які роблять його привабливим протоколом VPN як для звичайних користувачів, так і для мережевих адміністраторів. До цих ключових характеристик відносяться наступні:

Швидкий та ефективний: WireGuard спроектований як швидкий та ефективний протокол з мінімізацією навантаження на процесор і високою продуктивністю. Він може досягати такої самої високої швидкості, що й деякі застарілі та менш безпечні протоколи, забезпечуючи водночас покращену безпеку.

Безпечний: WireGuard використовує сучасну криптографію для забезпечення безпеки та конфіденційності зв'язку між вузлами мережі. Він використовує концепцію "Perfect Forward Secrecy" (PFS), що гарантує, що навіть при отриманні зловмисником приватного ключа, він не зможе розшифрувати попередні або майбутні спілкування.

Простота налаштування: WireGuard спроектований для простоти налаштування з читабельними та зрозумілими конфігураційними файлами. Він також підтримує аутентифікацію на основі ключів, що полегшує управління масштабними розгортаннями.

Крос-платформенний: WireGuard працює на різних операційних системах, таких як Linux, Windows, macOS, BSD, iOS та Android. Це робить його універсальним протоколом VPN, який можна використовувати в різних середовищах. WireGuard спроектований як простий та ефективний протокол з обмеженою базою коду та мінімізацією використання обчислювальних ресурсів. Він використовує UDP як транспортний протокол, що робить його менш чутливим до мережевого перевантаження та забезпечує ефективну роботу навіть у мережах із високими затримками. WireGuard взаємодіє з ядром операційної системи, що знаходиться ближче до апаратного забезпечення, ніж звичайні програми. Це головна причина, чому він здатний шифрувати та розшифровувати дані швидше.

WireGuard використовує ядерний простір, а OpenVPN - простір користувача. Це надає WireGuard перевагу в швидкодії через те, що у WireGuard немає необхідності постійно копіювати дані між різними частинами операційної системи, як це відбувається у OpenVPN. Крім того, WireGuard працює без постійного фонового сервісу, який потрібен для OpenVPN. Далі приведено приклад різниці роботи між протоколами від розробника Wireguard.

Protocol	Configuration
WireGuard	256-bit ChaCha20, 128-bit Poly1305
IPsec #1	256-bit ChaCha20, 128-bit Poly1305
IPsec #2	256-bit AES, 128-bit GCM
OpenVPN	256-bit AES, HMAC-SHA2-256, UDP mode

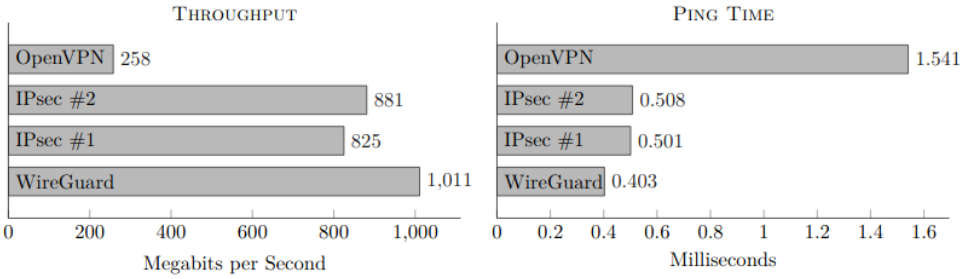

Рис.3.1 - Порівняння продуктивності різних VPN від Доненфельда

У обох тестах, які включали вимірювання пропускної здатності і часу відгуку пінгу, WireGuard показав значно кращі результати, ніж OpenVPN і два різновиди IPsec. Ці тести також показали, що використання OpenVPN і IPsec сильно навантажує центральний процесор, і в деяких випадках відбувається велика витрата ресурсів процесора. WireGuard, натомість, працює більш ефективно і не так сильно навантажує процесор, що дає можливість більш повною мірою використовувати можливості мережевої карти GigabitEthernet.

Звісно, можна припустити, що автор WireGuard може бути предвзятим у створенні тестових сценаріїв і тлумаченні результатів щодо продуктивності VPN-технологій.

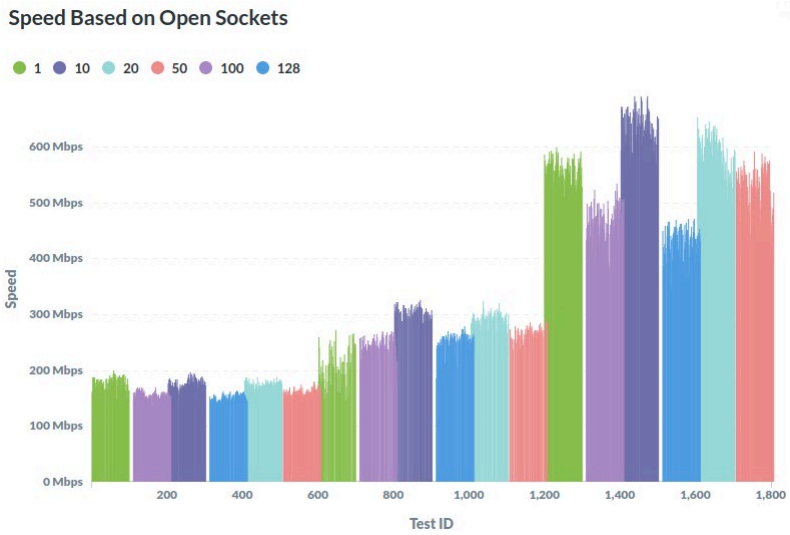

Рис.3.2 - Порівняння продуктивності OpenVPN та Wireguard від незалежного тестувальника.

Порівняння продуктивності WireGuard і OpenVPN в залежності від кількості відкритих сокетів. У тестах, які мають ідентифікатори в діапазоні від 0 до 600, використовувалися OpenVPN з UDP, від 700 до 1200 - OpenVPN з TCP, а від 1300 до 1800 - WireGuard.

Для автентифікації OpenVPN використовуються криптографічні методи, які ґрунтуються на бібліотеці OpenSSL. Ці методи пропонують різні способи захисту інформації, зокрема використовуючи шифрування, такі як AES, Blowfish, Camellia, ChaCha20, Poly1035, DES, Triple DES та інші. Також використовуються методи хешування, такі як MDS, MD4, SHA-1, SHA-2, BLAKE2 та інші для перевірки цілісності даних. Для створення безпечних з'єднань підтримуються різні методи генерації ключів, такі як RSA, DSA, X25519 та інші. При цьому ви можете вибрати один із двох протоколів для передачі даних: UDP або TCP.

WireGuard відрізняється від OpenVPN тим, що в ньому доступний обмежений набір криптографічних алгоритмів. Він використовує ChaCha20 для шифрування даних, Poly1035 для перевірки автентичності, BLAKE2s для обчислення хешів згідно із RFC7693, і SipHash24 для хешування ключів. Крім того, WireGuard підтримує як протокол передачі даних UDP, так і ідеальну пряму секретність.

Крім того, WireGuard використовує криптографію з відкритим ключем і має сувору автентифікацію. Він має захищений генератор ключів і автоматизує управління ключами. Попередній обмін ключами також підвищує безпеку. І навпаки, OpenVPN використовує сертифікати і приватний ключ для ідентифікації та шифрування.

WireGuard пропонує кращу швидкість і менший потенціал для порушень безпеки завдяки обмеженому вибору алгоритмів. OpenVPN, пропонуючи ширший спектр алгоритмів, має більш складні системи автентифікації та управління ключами.

OpenVPN використовує шифрування та автентифікацію на основі бібліотеки OpenSSL, яка має довгу історію і була широко протестована. З іншого боку, WireGuard використовує тільки власний тип шифрування ChaCha20 з автентифікацією Poly1305.

Крім того, OpenVPN використовує RSA і AES для своїх каналів передачі даних і управління, що знижує ризик атак злому паролів і ключів шифрування. Максимальна довжина ключа шифрування, яку підтримує OpenVPN, становить 4096 біт, тоді як WireGuard підтримує лише 256 біт. Наразі ні OpenVPN, ні WireGuard не мають суттєвих відомих вразливостей.

OpenVPN вважається безпечним вибором, але вимагає правильного налаштування. Його код був перевірений експертами та має високу підтримку. З іншого боку, код WireGuard легко перевіряється та на даний момент не має відомих вразливостей. Завдяки використанню сучасних криптографічних алгоритмів він є високозахищеним, швидким та складним для злому. У випадку виявлення вразливості, кінцеві точки негайно оновлюються до нової версії, що дозволяє уникнути використання скомпрометованого коду.

Для забезпечення захисту ваших даних найкращим вибором є OpenVPN. Він не зберігає жодних особистих даних своїх користувачів, таких як IP-адреси. З іншого боку, алгоритм WireGuard Cryptokey Routing тимчасово зберігає IP-адреси користувачів на VPN-сервері до перезапуску сервера. Однак наразі існують способи підвищити конфіденційність WireGuard.

WireGuard є відносно новим протоколом, але він може забезпечити таку ж безпеку, як і OpenVPN, який існує довше, пройшов більше сторонніх перевірок безпеки та має довший послужний список, ніж WireGuard. Зрілість WireGuard буде ще більш привабливою завдяки мінімальній кодовій базі та оновленим алгоритмам шифрування.

Консервативним вибором є OpenVPN. Однак WireGuard постійно вдосконалюється, впроваджуючи оновлені алгоритми шифрування. WireGuard є більш сучасним варіантом, що використовує новітні технології.

OpenVPN і WireGuard - чудові протоколи для VPN, але OpenVPN пропонує те, чого немає у WireGuard - можливість працювати через TCP. TCP є більш надійним, ніж UDP, тому він підходить для обходу суворих режимів цензури. Це пов'язано з тим, що TCP-порт 443 - це той самий порт, який використовує HTTPS. У цьому випадку OpenVPN залишається найбільш безпечним і стабільним. UDP

швидший і стабільніший при використанні з VPN-тунелями, але TCP є кращим для обходу цензури. Малоймовірно, що якась країна заблокує порт 443, який використовується для всіх основних видів діяльності, таких як електронна комерція і банківська справа. OpenVPN з протоколом TCP працює більш ефективно для обходу режимів цензури, ніж WireGuard.

В Linux існує стандартний підхід для створення зашифрованих тунелів, який використовує IPsec і рівень перетворення Linux. Користувачі можуть налаштовувати параметри ядра, визначаючи, які шифри, ключі або інші методи перетворення, такі як стиснення, використовувати для обробки пакетів, які проходять через IPsec. Зазвичай це відбувається через демон простору користувача, який оновлює ці налаштування на основі результатів обміну ключами, що здійснюється за допомогою протоколу IKEv2, який є складним і має широкий набір можливостей.

Однак, ця система має певні складності і значну кількість коду. Адміністратори також повинні мати розуміння окремої семантики брандмауера і безпечного маркування для пакетів, які працюють через IPsec. Відокремлення рівня обміну ключами від рівня транспортного шифрування або рівня перетворення є логічним з точки зору мережі, і також важливим щодо безпеки.

Однак цей підхід розшарування призводить до певної складності та ускладнює розгортання та реалізацію.

WireGuard вирішує проблему складності та розділення на рівні, яке існує в IPsec. Замість використання великої кількості налаштувань і рівнів, WireGuard надає простий віртуальний інтерфейс, наприклад, wg0, який можна налаштовувати за допомогою стандартних утиліт, таких як ip і ifconfig.

Після налаштування цього інтерфейсу з використанням приватних та відкритих ключів для безпечного обміну даними з одноранговими комп'ютерами, тунель починає працювати автоматично. Весь процес обміну ключами, установлення тунелів, розірвання з'єднань та інші дії відбуваються автоматично і надійно без необхідності втручання адміністратора.

Іншими словами, з точки зору адміністрування, інтерфейс WireGuard дуже простий. Правила брандмауера можна налаштовувати, використовуючи стандартну інфраструктуру для роботи з інтерфейсами брандмауера з гарантією того, що всі пакети, що надходять з інтерфейсу WireGuard, автентифікуються та зашифровуються автоматично.WireGuard є набагато менш схильним до катастрофічних збоїв і помилкової конфігурації порівняно з IPsec. Важливо відзначити, що IPsec має добре розроблене і надійне рівнеслідування, з високим рівнем академічної точності і правильності. Однак, із зростанням правильності на рівні абстракцій також збільшується складність використання, і досягнення повної надійності може бути важким завданням. З іншого боку, WireGuard виходить за межі збільшення складності, пов'язаної з багаторівневістю, і намагається вирішити проблеми, які виникають через це, за допомогою практичних інженерних рішень і криптографічних методів, які спрощують реалізацію та вирішують реальні проблеми безпеки.

З іншого боку спектра ми маємо OpenVPN, що базується на TUN/TAP та використовує TLS. Оскільки OpenVPN працює у користувацькому просторі, він має низьку продуктивність, оскільки пакети повинні копіюватися кілька разів між простором ядра та простором користувача. Крім того, OpenVPN вимагає постійно працюючого демона, і його адміністрування не так просто. Хоча інтерфейси TUN/TAP, такі як tun0, можуть мати подібні переваги до інтерфейсу wg0, як описано вище, OpenVPN все ж важко підтримувати через велику кількість коду, пов'язаного з TLS, що може призвести до потенційних вразливостей.

Для обміну ключами WireGuard вдихається прикладом OpenSSH, який використовує досить простий метод управління ключами. У нього є різні позасмугові механізми, і зазвичай рівноправні сторони обмінюються своїми статичними відкритими ключами. Це може бути таким простим, як обмін підписаними PGP-листами, або більш складним механізмом, як використання LDAP або центрів сертифікації. Важливо, що OpenSSH залишається агностичним стосовно механізму розподілу ключів.

WireGuard слідує подібному підходу. Два рівноправні учасники WireGuard обмінюються своїми відкритими ключами через деякий невизначений механізм, після чого вони можуть почати безпечну комунікацію. Другими словами, WireGuard не нав'язує жодного конкретного механізму розподілу ключів і залишає цю відповідальність на рівні користувача. Ще однією перевагою є те, що відкриті ключі у WireGuard дуже короткі, всього 32 байти, і можуть легко кодуватися у форматі Base64 у 44 символи, що робить їх зручними для передачі через різні канали.

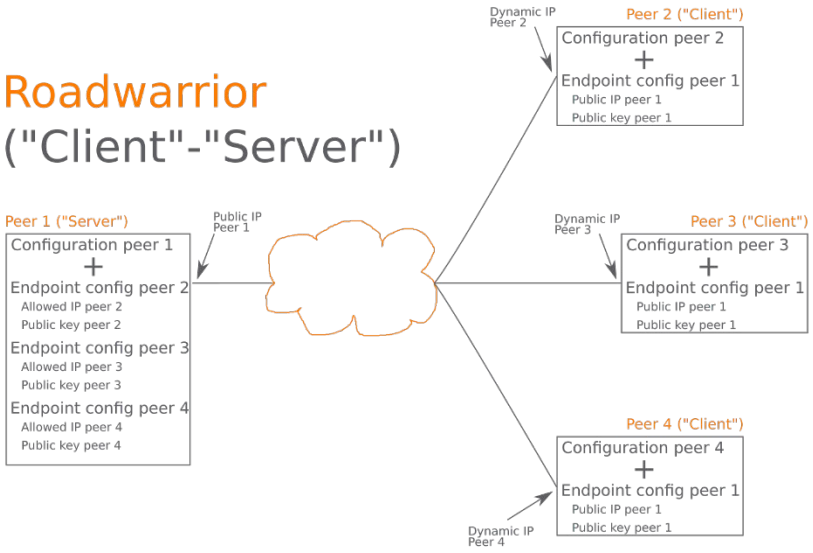

Рис.3.3 – Приклад структури клієнт-сервер Wireguard

WireGuard фокусується на роботі на третьому рівні мережі, що вважається найчистішим підходом для забезпечення автентифікації та атрибутивності пакетів. Розробники WireGuard вважають, що використання рівня 3 для об'єднання різних IP-мереж дозволяє спростити протокол, зробити його чистішим та легшим у реалізації. WireGuard підтримує рівень 3 як для IPv4, так і для IPv6, і може інкапсулювати пакети IPv4 у IPv6 та навпаки.

Одним з основних принципів безпечних VPN є зв'язок між одноранговими пристроями і набором IP-адрес, які кожен з них може використовувати для відправки даних. У WireGuard цей зв'язок встановлюється шляхом ідентифікації одноранговых вузлів за їхніми унікальними відкритими ключами, які складаються з 32 байтів і представляють собою точки на кривій Curve25519. Це означає, що існує

простий спосіб встановлення відповідності між відкритими ключами і набором дозволених IP-адрес. Сам інтерфейс WireGuard має свій приватний ключ і слухає на певному UDP-порту. Кожен одноранговий вузол ідентифікується за допомогою свого унікального відкритого ключа та має свій список дозволених IP-адрес джерела. Приклад таблиці на рис. 3.4

Configuration 1a

Interface Public Key	**Interface Private Key**	**Listening UDP Port**
HIgo...8ykw	yAnz..fBmk	41414
Peer Public Key	**Allowed Source IPs**	
xTIB...p8Dg	10.192.122.3/32, 10.192.124.0/24	
TrMv...WXX0	10.192.122.4/32, 192.168.0.0/16	
gN65..z6EA	10.10.10.230/32	

Рис.3.4 – Таблиця маршрутизації криптографічних ключів сервера

Сам інтерфейс WireGuard має свій приватний ключ і слухає на певному UDP-порту. Кожен одноранговий вузол ідентифікується за допомогою свого унікального відкритого ключа та має свій список дозволених IP-адрес джерела.

При передачі вихідного пакету через інтерфейс WireGuard, наприклад, wg0, використовується таблиця для визначення, який відкритий ключ слід використовувати для шифрування. Наприклад, якщо ми маємо пакет із IP-адресою призначення 10.192.122.4, цей пакет буде зашифровано з використанням відповідного відкритого ключа, наприклад, TrMv...WXX0.

Та навпаки, коли інтерфейс wg0 отримує зашифрований пакет, після його розшифрування і перевірки автентичності, він прийме пакет тільки у випадку, якщо відповідний запис про IP-адресу джерела був знайдений у таблиці і пов'язаний з відповідним відкритим ключем, який використовувався для розшифрування. Наприклад, якщо пакет був розшифрований за допомогою ключа xTIB...qp8D, він буде прийнятий лише у випадку, якщо IP-адреса джерела вказує на одну з діапазонів, які пов'язані з цим ключем (наприклад, від 10.192.122.3 або в діапазоні від 10.192.124.0 до 10.192.124.255); інакше пакет буде відкинуто. Завдяки цьому простому принципу адміністратори можуть легко користуватися простими правилами брандмауера. Наприклад, вхідний пакет на інтерфейсі wg0 з IP-адресою

джерела 10.10.10.230 може бути надійно визнаним як пакет від однорангового пристрою з відповідним відкритим ключем, наприклад, gN65...Bz6E.

У загальному випадку всі пакети, які надходять на інтерфейс WireGuard, мають достовірну IP-адресу джерела. Це можливо завдяки тому, що WireGuard працює виключно на третьому рівні мережі. На відміну від деяких інших поширених протоколів VPN, як, наприклад, L2TP/IPsec, використання аутентифікованої ідентифікації однорангових пристроїв на рівні 3-го рівня сприяє більш прозорому та ефективному дизайну мережі.

Якщо один одноранговий пристрій WireGuard бажає направляти весь свій трафік через інший одноранговий пристрій WireGuard, налаштування таблиці маршрутизації криптографічних ключів може бути спрощене. Приклад таблиці на рис. 3.5

Configuration 2a

Interface Public Key	**Interface Private Key**	**Listening UDP Port**
gN65..z6EA	gI6E..fWGE	21841
Peer Public Key	**Allowed Source IPs**	
HIgo..8ykw	0.0.0.0/0	

Рис.3.5 – Таблиця маршрутизації криптографічних ключів клієнта

В даному випадку одноранговий пристрій дозволяє HIgo...f8yk надсилати пакети на інтерфейс wg0 з будь-якою вихідною IP-адресою, і всі ці пакети, які виходять з інтерфейсу wg0, будуть зашифровані за допомогою захищеного сеансу, що пов'язаний із відповідним відкритим ключем, і відправлені до цього однорангового вузла.

Звісно, важливо, щоб однорангові пристрої могли надсилати зашифровані UDP-пакети WireGuard один одному на відомі зовнішні IP-адреси та UDP-порти в Інтернеті. Кожен одноранговий пристрій в таблиці маршрутизації криптографічного ключа може передбачено вказати ці зовнішні IP-адреси та UDP-порти для кожної конкретної кінцевої точки іншого однорангового пристрою. Важливо відзначити, що це поле є необов'язковим, і якщо воно не вказано, WireGuard використовуватиме зовнішню IP-адресу джерела для визначення кінцевої точки автоматично.

Важливо відзначити, що порт прослуховування однорангових вузлів і порт джерела відправлених пакетів завжди однакові, що значно спрощує конфігурацію та забезпечує надійний прохід через NAT. Ця властивість роумінгу також гарантує, що однорангові пристрої завжди мають актуальні зовнішні IP-адреси та UDP-порти, і, таким чином, не потрібно підтримувати відкриті сеанси NAT протягом тривалого часу.

У випадках, коли необхідно тримати сеанс NAT або брандмауера відкритим протягом тривалого часу, інтерфейс можна додатково налаштувати для періодичного відправлення постійних автентифікованих пакетів.

Така архітектура забезпечує високий рівень зручності і вимагає мінімальної конфігурації. Хоча зловмисники з атакою "людини посередині" можуть спробувати модифікувати ці зовнішні неавторизовані IP-адреси, але вони не матимуть можливості розшифрувати або змінити корисне навантаження пакетів.

Для ініціації надсилання зашифрованих інкапсульованих пакетів потрібно виконати процес рукостискання обміну ключами, відомий як 1-RTT (one round trip time). Ініціатор починає, надсилаючи повідомлення відповідачу, і відповідач відповідає на це повідомлення ініціатору. Після завершення цього процесу рукостискання ініціатор може почати надсилати зашифровані повідомлення, використовуючи спільну пару симетричних ключів: один для відправки інформації, інший для отримання інформації відповідачем. Після надсилання першого зашифрованого повідомлення від ініціатора до відповідача, відповідач може почати надсилати зашифровані повідомлення ініціатору.

Це послідовність обмежень порядку, які були введені з метою вимагати підтвердження, зокрема таке, як описано в KEA+C. Крім того, це дозволяє обробляти повідомлення рукостискання асинхронно для подальшої передачі даних. Всі ці повідомлення використовують шаблон "IK" від Noise, додатково додаючи новий конструкт файлу "cookie" для захисту від атак на відмову в обслуговуванні. У результаті отримуємо дуже надійну систему безпеки, яка відповідає вимогам безпеки обміну автентифікованими ключами (AKE). Вона уникає імітації компрометації ключа, атак на відтворення, забезпечує ідеальну пряму секретність,

ідентифікацію приховування статичних відкритих ключів, подібно до SIGMA, і має стійкість до атак на відмову в обслуговуванні.

WireGuard має за мету уникати збереження будь-якого стану до автентифікації та не відправляти жодних відповідей на неавтентифіковані пакети. Неавтентифіковані пакети не викликають збереження стану і не генерують жодних відповідей, що робить WireGuard незрозумілим для нелегітимних однорангових пристроїв та мережевих сканерів.

WireGuard покладається на той факт, що автентифікація повинна відбуватися в першому пакеті, отриманому від відповідача. Проте ця властивість може відкривати деякий потенційний ризик атаки на відтворення. Зловмисник може спробувати відтворити початкові повідомлення рукостискання, щоб обдурити відповідача та змусити його відновити ефемерний ключ ініціатора. Це може призвести до недійсності сеансу законного ініціатора, але не вплине на секретність або автентичність будь-яких повідомлень.

Така архітектура дозволяє уникнути динамічного виділення пам'яті навіть для автентифікованих пакетів, що робить WireGuard дуже ефективним та мінімізує його стан.

Для запобігання атакам на відтворення, перше повідомлення включає 12-байтову мітку часу TAI64N, яка шифрується та аутентифікується. Відповідач відслідковує найбільшу отриману часову мітку від однорангового партнера і відкидає пакети, які містять часові мітки, менші або рівні цій найбільшій мітці часу. Ця мітка часу не обов'язково повинна бути точною, але повинна бути монотонно зростаючим 96-бітним числом, що призначено для однорангового вузла.

Навіть якщо відповідач втратить цей стан через перезапуск, це не становить проблему. Перезапуск відповідача не вплине на поточні захищені сесії, оскільки в цей момент відповідач не має активних захищених сесій, які можна було б порушити. Після того, як ініціатор відновить захищений сеанс з відповідачем після його перезапуску, ініціатор буде використовувати більшу мітку часу, що скасує попередню. Ця мітка часу гарантує, що зловмисники не зможуть порушити поточну сесію між ініціатором і відповідачем за допомогою атаки на повтор. Це також

означає, що два різні однорангових пристрої не повинні мати спільних приватних ключів, оскільки в такому випадку пакет, надісланий одному з них, може бути відтворений іншим, і подальша відповідь змусить ініціатора навмисно переключатися від одного однорангового вузла до іншого.

Технічно TAI64N є зручним у реалізації через свій big-endian (спосіб зберігання байтів у пам'яті комп'ютера) формат, що спрощує порівняння двох 12-байтових міток часу за допомогою стандартної функції memcmp(). Під час створення ініціаційного повідомлення, WireGuard вірить лише результатам обміну Діффі-Хеллмана статичними ключами обох сторін, які беруть участь у процедурі автентифікації.

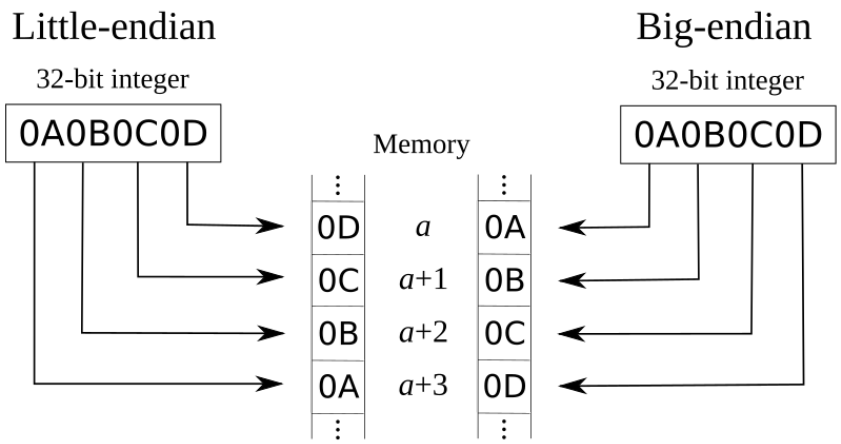

Рис.3.6 – Порівняння способів зберігання байтів в памяті комп'ютера

Це означає, що навіть якщо один із статичних ключів буде порушений, зловмисник може спробувати підробити ініціаційне повідомлення, але йому не вдасться завершити рукостискання, оскільки цей процес містить максимальну мітку часу. Ця властивість перешкоджає успішному підключенню у майбутньому в разі компрометації ключа. WireGuard цим різниться від традиційних уразливостей імітації компрометації ключа, до яких він не піддається.

У випадку, якщо точність мітки часу TIA64N породжує небажаний ризик витоку інформації, реалізації можуть вирішити це питання, обрізавши 24-бітну наносекундну частину мітки часу.

WireGuard ґрунтується на ідеї обміну статичними відкритими ключами між одноранговими системами перед передачею даних. Ці статичні ключі виступають

як статичні ідентифікатори кожного вузла і є основою для забезпечення безпеки всієї комунікації.

Всі дані, які передаються між вузлами, залишаються секретними завдяки безпеці функції Curve25519 ECDH (англ. Elliptic Curve Diffie-Hellman), яка використовується для обміну ключами та шифрування даних. Ця функція забезпечує високий рівень секретності та важко піддається атакам.

Зокрема, для захисту від майбутніх прогресів у квантових обчисленнях, WireGuard має можливість додатково розділити один симетричний ключ шифрування між парою однорангових вузлів. Це надає додатковий шар симетричного шифрування для підвищення безпеки.

В моделі атаки передбачається, що зловмисники можуть записувати зашифрований трафік протягом тривалого періоду з метою спроб розшифрування його у майбутньому.

Під час попереднього обміну симетричними ключами шифрування може виникнути проблема управління ключами та підвищеного ризику компрометації. Принциповою ідеєю WireGuard є використання попередньо наданих симетричних ключів, які діють як додатковий шар безпеки.

Ця ідея полягає в тому, що на момент, коли квантові обчислення, які можуть потенційно зламати Curve25519, стануть реальністю, попередньо відкритий симетричний ключ вже буде застарілим та неактуальним. Це означає, що досить ймовірно, що ніхто не матиме доступу до цього ключа на той час.

Навіть якщо попередньо відкритий симетричний ключ стане відомий, ключі Curve25519 все ще забезпечують надзвичайний рівень безпеки в найближчій перспективі. Цей гібридний підхід, що об'єднує симетричне шифрування з попереднім обміном ключів і криптографію еліптичної кривої, забезпечує надійний компроміс для тих, хто параноїчно ставиться до безпеки.

Цей підхід також дозволяє створювати складні схеми ротації ключів на основі WireGuard для досягнення різних рівнів безпеки після можливої компрометації.

Обчислення множення точок на кривій Curve25519 може вимагати значних обчислювальних ресурсів, навіть при високій швидкості обробки цієї кривої на більшості сучасних процесорах. Перевірка автентичності повідомлення рукостискання вимагає виконання операцій множення на кривій Curve25519, що створює потенційну можливість для атак на відмову в обслуговуванні.

Для того, щоб захистити себе від можливих атак на виснаження процесорних ресурсів, відповідач, який є отримувачем повідомлення рукостискання, може вирішити не обробляти саме це повідомлення (як і початкове, так і відповідь на нього). Замість цього він може надіслати відповідь, яка містить щось, що називається "файлом cookie."

Надалі, ініціатор може використовувати цей файл cookie для повторної відправки повідомлення рукостискання у майбутньому. Це дозволяє впевнитися, що відповідач обробить повідомлення, оскільки воно містить дійсний файл cookie, і відбудеться успішний обмін ключами без ризику атаки на виснаження процесорних ресурсів.

Відповідач підтримує безпеку шляхом використання секретного випадкового значення, яке періодично змінюється, наприклад, кожні дві хвилини. Файл cookie, в цьому контексті, представляє собою результат обчислення MAC-адреси на основі IP-адреси ініціатора за допомогою цього змінюваного секретного ключа.

При відправці свого повідомлення ініціатор також включає MAC-адресу, обчислену на основі своєї IP-адреси і використовуючи цей файл cookie як ключ. Коли відповідач отримує таке повідомлення, він перевіряє, чи правильна MAC-адреса вказана в повідомленні, використовуючи файл cookie як ключ для порівняння.

Ця система дозволяє відповідачу перевірити відповідність між повідомленнями та IP-адресами, що їх надсилають, і таким чином підтвердити право на використання конкретної IP-адреси. Це також може бути використано для обмеження швидкості передачі даних шляхом застосування класичних алгоритмів обмеження швидкості на основі IP-адрес.

Ця схема схожа на ту, що використовують DTLS та IKEv2, але має кілька суттєвих недоліків:

По-перше, Wireguard вважає за краще дотримуватися стратегії мовчання, не висилати жодних відповідей на неавтентифіковані повідомлення. Включення невибіркового надсилання повідомлень-відповідей з файлом cookie може порушити цю принципову властивість.

По-друге, надсилання файлу cookie відкритим текстом не є безпечним, оскільки зловмисники можуть використовувати його для надсилання шахрайських повідомлень, які можуть бути прийняті і оброблені.

По-третє, навіть сам ініціатор може стати жертвою атаки на відмову в обслуговуванні, надсилаючи шахрайські файли cookie, які в подальшому можуть бути безрезультатно використані для обчислення MAC-адреси його повідомлень.

Щоб впоратися з першою проблемою і забезпечити тишу відповідача, навіть під навантаженням, введено вимогу до всіх повідомлень мати перший MAC (msg.mac1). Цей MAC використовує відкритий ключ відповідача і, в принципі, вимагає, щоб користувач, який надсилає повідомлення, знав, з ким він спілкується (оскільки це передбачає знання відкритого ключа відповідача), щоб отримати відповідь. Незалежно від навантаження, цей перший MAC (msg.mac1) повинен завжди присутнім і дійсним.

Хоча відкритий ключ відповідача не є суворим секретом, в рамках цієї моделі атаки він є досить "полусекретним", оскільки він визначає наявність об'єкта для спілкування. До певної міри, це забезпечує приховування факту існування об'єкта спілкування. Необхідність знання відкритого ключа відповідача може допомогти зловмисникові припущати, для кого призначене повідомлення, хоча це припущення не є криптографічним доказом, оскільки воно не використовує жодного конфіденційного матеріалу при створенні MAC-адреси.

Для подолання другої проблеми - передачі MAC-адреси відкритим текстом, ми застосовуємо схему AEAD з розширеним рандомізованим нонсом до кукі під час передачі. Ця схема використовує в якості симетричного ключа шифрування публічний ключ відповідача. Ще раз, враховуючи нашу загальну модель атаки, в

більшості випадків публічне значення ключа відповідача вважається досить "полусекретним" і вистачає для вирішення проблеми, пов'язаної з відкритим текстом MAC-адреси під час передачі.

Зрештою, щоб вирішити третю проблему, ми застосовуємо поле "додаткові дані" в AEAD для шифрування куки. Це додаткова захисна міра для автентифікації першого MAC-адреси (msg.mac1) ініціюючого повідомлення, яке призвело до створення файла cookie у відповідь. Це гарантує, що зловмисники, які не перебувають в позиції "людини посередині", не можуть відправляти недійсні відповіді на файли cookie ініціаторам, щоб завадити їхній автентифікації за допомогою правильного файла cookie. Таким чином, зловмисник, який займає пасивну позицію "людини посередині", може спробувати відкинути відповіді на файли cookie для завади з'єднанню, але цей випадок не є дійсно актуальним. Зловмисник, який виступає як "людина посередині", може спростувати ці пакети, але це вже менш проблематично, і це практично не відрізняється від атаки на відмову в обслуговуванні в TCP. Іншими словами, ми використовуємо поле AD для прив'язки відповідей на файли cookie до ініційних повідомлень. Розв'язавши ці проблеми, ми можемо додати другий MAC (msg.mac2), використовуючи файл cookie як безпечний MAC-ключ. Коли відповідач перебуває під навантаженням, він буде приймати лише ті повідомлення, які додатково мають цей другий MAC. Після встановлення початкового захищеного сеансу WireGuard спробує створити новий сеанс, надсилаючи ініціююче повідомлення рукостискання після передачі певної кількості повідомлень транспортних даних, визначеної як Rekey-After-Messages. Також, якщо одноранговий вузол ініціює поточний безпечний сеанс, WireGuard надсилає ініціююче повідомлення рукостискання для розпочатку нового безпечного сеансу у випадку, якщо час, який пройшов після передачі повідомлень транспортних даних поточного безпечного сеансу, перевищує Rekey-After-Time в секундах, або якщо час, який пройшов після отримання повідомлення транспортних даних поточного безпечного сеансу, перевищує (Reject-After-Time - Keepalive-Timeout - Rekey-Timeout) секунд, і у цей час поточний безпечний сеанс не використовується. Ця можливість для оппортуністичної зміни ключа,

ініційована ініціатором поточного сеансу, призначена для уникнення проблеми "гримучого стада", коли обидва вузли намагаються створити новий сеанс одночасно. Завдяки пасивній функції підтримки активності, ініціація, викликана старим безпечним сеансом, зазвичай досить для забезпечення створення нових сеансів, які створюються кожні Rekey-After-Time секунди. Однак для випадків, коли одноранговий вузол отримав дані, але не має жодних даних для надсилання негайно, і час до закінчення другого строку відмови (Reject-After-Time) закінчився раніше, ніж Keepalive-Timeout секунд, тоді ініціація, викликана застарілим безпечним сеансом, відбувається при настанні першого випадку перед настанням останнього. Після виконання Rekey-After-Messages або в разі закінчення часу для відмови поточного безпечного сеансу, визначеного в секундах (Reject-After-Time), WireGuard відмовляється надсилати або приймати будь-які додаткові транспортні дані повідомлень, використовуючи поточний безпечний сеанс, доки новий безпечний сеанс не буде створено через рукостискання 1-RTT.

Нові безпечні сесії створюються приблизно кожні Rekey-After-Time секунд (що ймовірніше стається до передачі даних через Rekey-After-Messages). Це означає, що безпечна сесія постійно ротується, створюючи новий тимчасовий симетричний ключ кожного разу для забезпечення ідеальної перспективної секретності. Але слід мати на увазі, що після отримання ініціатором відповіді на рукостискання відповідач не може відправити транспортні дані до тих пір, поки не отримає перше повідомлення з транспортними даними від ініціатора. Крім того, транспортні дані, зашифровані за допомогою попередньої безпечної сесії, можуть перебувати в транзиті після створення нової безпечної сесії. З цих причин WireGuard зберігає в пам'яті поточну безпечну сесію, попередню безпечну сесію та наступну безпечну сесію для непідтвердженої сесії. Кожного разу, коли створюється нова безпечна сесія, існуюча переміщується в "попередній" слот, а нова займає "поточний" слот для ініціатора, а для відповідача слот "наступний" використовується до підтвердження рукостискання. "Попередньо-попередня" сесія потім видаляється, а її пам'ять обнуляється. Якщо протягом (Reject-After-Time × 3) секунд не створено нову безпечну сесію, поточну безпечну сесію, попередню

безпечну сесію і, можливо, наступну безпечну сесію видаляють і обнулюють, а також будь-які можливі частково завершені стани рукостискання та тимчасові ключі. Коли користувач вперше надсилає пакет через інтерфейс WireGuard, пакет не може негайно бути відправлений, оскільки жодна поточна сесія не існує. Таким чином, після встановлення пакета в чергу, WireGuard відправляє повідомлення ініціації рукостискання. Після відправлення повідомлення ініціації рукостискання через певний час, з урахуванням умови першого пакета, якщо відповідь на рукостискання не надходить протягом Rekey-Timeout секунд, конструюється і відправляється нове повідомлення ініціації рукостискання з новими випадковими тимчасовими ключами. Ця повторна ініціація виконується протягом Rekey-Attempt-Time секунд перед тим, як відмовитися, хоча цей лічильник скидається, коли пірі ситуативно спробує відправити нове повідомлення з транспортними даними. Критично важливою майбутньою роботою є налаштування значення Rekey-Timeout для використання експоненційного відстрочення, замість поточного фіксованого значення. WireGuard впроваджує пасивний механізм утримання для забезпечення активності сесій та можливості для обох сторін пасивно визначати, чи відбулася помилка чи відключення з'єднання. Якщо пір отримав достовірне аутентифіковане повідомлення з транспортними даними, але сам не має пакетів для відправки протягом Keepalive-Timeout секунд, він відправляє повідомлення про утримання. Повідомлення про утримання просто є повідомленням з транспортними даними із вкладеним зашифрованим внутрішнім пакетом нульової довжини. Оскільки всі інші повідомлення з транспортними даними містять IP-пакети, які мають мінімальну довжину, це повідомлення про утримання може бути легко визначене за допомогою того, що воно має нульову довжину вкладеного пакета. Це пасивне утримання відправляється тільки тоді, коли пір нічого не має для відправлення, і це відправляється лише у випадках, коли інший пір відправляє аутентифіковані повідомлення з транспортними даними. Це означає, що, коли жодна зі сторін не обмінюється повідомленнями з транспортними даними, мережеве з'єднання буде мовчазним. Оскільки кожне відправлене повідомлення з транспортними даними вимагає відповіді якогось виду - або органічної, створеної

за природою вкладених пакетів, або цього повідомлення утримання - ми можемо визначити, чи відбулася помилка або розірване з'єднання безпеки, якщо повідомлення з транспортними даними не було отримано протягом (Keepalive-Timeout + Rekey-Timeout) секунд. У цьому випадку ініціюється повідомлення ініціації рукостискання для неактивного піра, один раз кожні Rekey-Timeout секунд, доки безпечна сесія успішно не створиться або доки не мине Rekey-Attempt-Time секунд. Коли одна зі сторін перенавантажена, ініціаційне повідомлення, рукостискання чи відповідь на рукостискання може бути відхилено, і надіслано відповідь зі значенням "cookie". Після отримання повідомлення з відповіддю cookie, що дозволяє піру надіслати нове ініційне чи відповідне повідомлення з дійсним msg.mac2, яке не буде відхилено, пір не повинен негайно відправляти тепер валідне повідомлення. Замість цього він просто повинен зберегти розшифроване значення cookie з повідомлення з відповіддю cookie та чекати, поки таймер Rekey-Timeout не виконається для повторної спроби ініціювання рукостискання. Це запобігає можливому зловживанню генерації пропускної здатності та сприяє зменшенню умов навантаження, які вимагають повідомлення з відповіддю cookie в першу чергу.

Реалізація WireGuard в ядрі Linux має кілька цілей. По-перше, вона повинна бути короткою і простою, щоб перевірка та рецензування коду на наявність вразливостей в області безпеки були не лише простими, але й приносили задоволення; WireGuard реалізований у менше ніж 4 000 рядках коду (за винятком криптографічних примітивів). По-друге, він повинен бути надзвичайно швидким, щоб конкурувати за продуктивністю з IPsec. По-третє, він повинен уникати виділення пам'яті та інших ресурсів відповідно до вхідних пакетів. По-четверте, він повинен інтегруватися якнайнативніше та безшовно з існуючою інфраструктурою ядра та очікуваннями користувача, інструментами та API. І, по-п'яте, його повинно бути можливо будувати як зовнішній модуль ядра без вимог до внесення змін до основного ядра Linux. WireGuard не є просто академічним проектом з кодом, що ніколи не випускався з лабораторії, але, скоріше, практичним проектом, який має на меті втілення готових до використання реалізацій. Драйвер пристрою WireGuard

має прапорці, які повідомляють ядру, що він підтримує загальне відокремлення сегментів (GSO), збір введення-виведення (scatter gather I/O) та апаратне відключення контрольної суми. Усе це означає, що ядро передає WireGuard "супер-пакети", які значно перевищують розмір MTU, такі які вже були передані верхніми рівнями, такими як TCP або системи затворення TCP та UDP. Це дозволяє WireGuard працювати з групами партій вихідних пакетів. Після розділення пакетів на частини розміром ≤MTU, WireGuard намагається зашифрувати, інкапсулювати і відправити через UDP всі їх одночасно, кешуючи інформацію маршрутизації, щоб обчислення відбувалося лише один раз для кластера пакетів. Це має дуже важливий ефект також на зменшення промахів кеша: очікуючи, доки всі індивідуальні пакети супер-пакета будуть зашифровані і відправлені, щоб передати їх мережевому рівню, складний і інтенсивний на процесор дійсний рівень тримає інструкції, проміжні змінні та передбачення гілок в кеші процесора, що в багатьох випадках призводить до збільшення продуктивності відправки на 35%. Іноді вихідні пакети повинні зачекати до успішного завершення рукостискання. Коли пакети нарешті можуть бути відправлені, весь ряд існуючих зачеканих пакетів трактується як один супер-пакет, щоб скористатися тими ж оптимізаціями, що і вище. Нарешті, для уникнення непотрібних виділень пам'яті, всі перетворення пакетів виконуються на місці, уникнення необхідності у копіюванні. Це стосується не лише шифрування та розшифрування даних, які відбуваються на місці, але також деяких даних користувача та файлів, які відправляються за допомогою sendfile; це обробляється цією системою чергування супер-пакетів без копіювання. В порівнянні з WireGuard, рівень xfrm має той плюс, що він не вимагає виконання криптографії у softirq, що робить його більш гнучким. Тим не менш, існує попередній приклад обробки криптографічних операцій у softirq на рівні інтерфейсу: підсистема mac802111, яка використовується для шифрування WPA в бездротових мережах. WireGuard, як віртуальний інтерфейс, який виконує шифрування, архітектурно не настільки відрізняється від бездротових інтерфейсів, які також виконують шифрування на цьому рівні. Хоча на практиці це дійсно добре працює, воно не є паралельним. З цієї причини використовується система padata ядра для

паралельного розподілу операцій шифрування та розшифрування серед конкуруючих операцій для використання всіх процесорів і ядер CPU. Також можна обчислювати контрольні суми пакетів паралельно з використанням цього методу. Однак при відправці пакети повинні бути відправлені впорядковано, що означає, що кожен пакет не може просто негайно відправлятися після того, як він зашифрований. На щастя, API padata розбиває операції на паралельний етап, за яким слідує послідовний послідовний етап. Це також корисно для паралельного розшифрування, де лічильник повідомлень повинен бути перевірений та збільшений в порядку прибуття пакетів, щоб їх не відхиляли непотрібно. З метою зменшення латентності, якщо є тільки один пакет у супер-пакеті і його довжина менше 256 байтів, або якщо є тільки одне активне ядро CPU, пакет обробляється у softirq. Так само ініціаційні та відповідні повідомлення рукостискання та повідомлення з відповіддю cookie обробляються на окремому паралельному низькопріоритетному робочому потоці. Операції ECDH витратні за ресурсами CPU, тому важливо, щоб потік роботи з рукостискання не монополізував CPU. Для цього використовуються низькопріоритетні фонові робочі черги для цієї асинхронної обробки повідомлень рукостискання. Для інтеграції з існуючими службами IP та простором користувача Linux, основний шар RTNL ядра використовується для реєстрації віртуального інтерфейсу, відомого в ядрі як "ланка". Це легко надає доступ до API ядра, доступного через ip-link та ip-set. Для конфігурування приватного ключа інтерфейсу та публічних ключів та кінцевих точок пір, спочатку було використано повідомлення RTM_SETLINK RTNL, але це виявилося занадто обмеженим. Виявилося, що набагато чистше просто реалізувати API на основі ioctl, передаючи серію структур туди і назад. Хоча цей підхід був досить чистим, стек мережі Linux рухається до конфігурації, що повністю базується на Netlink, тому в кінцевому підсумку був вибраний загальний протокол Netlink. Окремий інструмент простору користувача, wg, використовується для зв'язку через Netlink, і майбутні плани включають в себе інтеграцію цієї функціональності безпосередньо в ip. Хоча ядро Linux вже містить дві вдосконалені реалізації таблиць маршрутизації - LC-trie для IPv4 та радісний трай для IPv6 - вони тісно

пов'язані з рівнем маршрутизації FIB і зовсім не придатні для інших цілей. З цієї причини була розроблена дуже мінімальна таблиця маршрутизації. Автори досягли успіху в реалізації таблиці маршрутизації cryptokey як таблиці розподілу лотереї, LC-trie та стандартного радісного трай, кожен з яких надавав достатньо, але трохи відрізнявся характеристиками продуктивності. У кінцевому підсумку було обрано простоту радісного трай, який має хороші характеристики продуктивності та можливість реалізувати його з безблокуючими пошуками за допомогою системи RCU. Кожного разу, коли вихідний пакет проходить через WireGuard, призначений пір перевіряється за допомогою цієї таблиці, і кожного разу, коли вхідний пакет досягає WireGuard, його дійсність перевіряється за допомогою цієї таблиці, тому тут дійсно важлива продуктивність. Для всіх повідомлень ініціювання рукостискання респондент повинен здійснити пошук розшифрованого статичного відкритого ключа ініціатора. Для цього WireGuard використовує хеш-таблицю, використовуючи надзвичайно швидку функцію SipHash2-4 MAC з секретом, так щоб верхні рівні, які можуть забезпечувати інтерфейс WireGuard відкритими ключами в більш складній схемі розподілу ключів, не могли здійснити атаку на відмову обслуговування через колізії хеш-таблиці. Хоча криптографічний API ядра Linux має велику колекцію примітивів і призначений для повторного використання в різних системах, API вводить зайву складність та виділення пам'яті. Кілька ревізій WireGuard використовували криптографічне API з різними техніками інтеграції, але в кінцевому підсумку виявилося, що використання примітивів напряму з прямими, неабстрагованими API виявилось набагато чистішим і менш витратним за ресурсами. Як стек, так і тиск купи були зменшені завдяки прямому використанню криптографічних примітивів, а не через криптографічне API ядра. Криптографічне API також ускладнює уникнення виділень пам'яті при використанні кількох ключів у багатогранних способах, необхідних для протоколу Noise. На момент написання WireGuard має оптимізовані реалізації ChaCha20Poly1305 для різних розширень векторних операцій архітектури Intel, з реалізаціями для ARM/NEON та MIPS на шляху. Найшвидша реалізація, підтримувана апаратурою, вибирається під час виконання, із використанням

числового процесора оптимістично. Всі ефемерні ключі та проміжні результати криптографічних операцій після використання обнуляються з пам'яті для забезпечення ідеальної передбачуваності вперед та запобігання можливим витокам інформації. Компілятор повинен бути спеціально повідомлений про це явне обнулення, щоб "мертвий залишок" не був оптимізований, і для цього ядро надає функцію memzero_explicit. На відміну від криптографічних примітивів, існуючі реалізації ядра мережі токен-бакету з обмеженням швидкості на основі хеш-функцій, для обмеження швидкості повідомлень ініціювання рукостискання та відповіді при перевантаженні після призначення IP cookie, були дуже мінімальними та легкими для повторного використання в WireGuard. WireGuard використовує зіставник хешу Netfilter для цього. На менше ніж 4 000 рядках коду WireGuard демонструє, що можливо створити безпечні мережеві тунелі, які реалізовані просто, вкрай продуктивні, використовують сучасну криптографію і залишаються легкими для адміністрування. Простота дозволяє легко провести незалежну верифікацію та реімплементацію на різноманітті платформ. Використані криптографічні конструкції та примітиви забезпечують високу швидкість на різноманітті пристроїв, від серверів дата-центрів до мобільних телефонів, а також надійні властивості безпеки в подальшому. Легкість впровадження також усуне багато звичайних та руйнівних помилок, які зараз бачимо у багатьох випадках впровадження IPsec. У відміну від цього, WireGuard акцентується на простоті та зручності використання, при цьому надаючи масштабовану і високо безпечну систему. Будучи беззвучним для ненадійних пакетів і не виділяючи пам'ять для великої частини ресурсів, він може бути впроваджений на віддалених краях мережі як надійна точка доступу, яка не легко виявляється для атак і не надає важливої мішені для атак. Парадигма криптографічної маршрутизації ключів легка для вивчення і сприятиме безпечному проектуванню мереж. Протокол ґрунтується на криптографічно обґрунтованих та консервативних принципах, використовуючи добре зрозумілі, але сучасні криптопримітиви. WireGuard був розроблений з практичної точки зору, призначений для вирішення реальних проблем безпеки мереж.

3.2 Застосування Wireguard та Ansible у сучасних інформаційних системах

Wireguard і Ansible - це два потужних інструменти, які можуть взаємодіяти для створення безпечних та автоматизованих мережевих рішень, які відповідають вимогам сучасних інформаційних систем. Wireguard, як VPN-протокол, надає безпечні тунелі для з'єднання мереж і приватних систем через ненадійні мережі, такі як Інтернет. Ansible, з іншого боку, є інструментом для автоматизації конфігурації та управління інфраструктурою, який дозволяє використовувати інфраструктуру як код. Спільне використання цих двох інструментів відкриває двері для створення ефективних, безпечних і автоматизованих мережевих рішень.

Wireguard забезпечує безпеку шляхом використання сучасних криптографічних методів і мінімізації атакованих поверхонь. Однак ручне налаштування і управління багатьма серверами і клієнтами може бути складним та часомістким завданням, особливо в великих мережевих середовищах. Тут вступає в гру Ansible. Ansible дозволяє створювати автоматизовані задачі для налаштування Wireguard-тунелів, що робить процес створення та управління безпечними VPN-з'єднаннями більш ефективним і менш вразливим до помилок адміністраторів.

Один із способів, яким Wireguard і Ansible можуть взаємодіяти, полягає в автоматизованому розгортанні VPN-тунелів. Ansible може створити і встановити конфігураційні файли для Wireguard на серверах та клієнтах, а також забезпечити налаштування правил брандмауера та маршрутизації. Це важливо, оскільки неправильне налаштування може призвести до уразливостей у мережевому захисті. Ansible дозволяє систематизувати і контролювати цей процес, що робить його більш надійним і менш чутливим до помилок.

Крім того, Ansible дозволяє виконувати централізоване керування VPN-мережею. Інформаційні системи можуть містити багато серверів та клієнтів, і керування ними вручну може бути незручним і витратним. Ansible дозволяє визначити стан мережі та виконувати зміни в ній шляхом визначення задач у вигляді "інфраструктури як коду". Це означає, що можна описати бажаний стан

мережі у конфігураційних файлах, і Ansible автоматично намагатиметься досягнути цього стану, надсилаючи необхідні команди на сервери та клієнти Wireguard. Такий підхід дозволяє легко масштабувати мережу та забезпечувати її сталу послугу.

Важливою частиною безпеки мережі є її постійне моніторинг та оновлення. Ansible може бути використаний для автоматичного виявлення помилок і вразливостей у мережі, а також для розгортання оновлень та змін в конфігураціях Wireguard. Це допомагає забезпечити актуальність захисних заходів і знижує ризик вразливостей через застарілу або некоректну конфігурацію.

Додатковою перевагою є можливість створення бекапів конфігурацій Wireguard за допомогою Ansible. Це важливо для відновлення робочого стану мережі після аварій або випадків втрати даних. Ansible дозволяє створювати резервні копії конфігурацій і відновлювати їх швидко та ефективно.

В цілому, використання Ansible для налаштування та адміністрування серверу Wireguard робить процес більш ефективним та безпечним, дозволяючи адміністраторам легко керувати інфраструктурою VPN та забезпечити її безпеку та стабільність.

4 ВПРОВАДЖЕННЯ МЕТОДУ ВІДДАЛЕНОГО АДМІНІСТРУВАННЯ СЕРВЕРУ WIREGUARD ЗА ДОПОМОГОЮ СИСТЕМИ УПРАВЛІННЯ КОНФІГУРАЦІЯМИ ANSIBLE

4.1 Встановлення та налаштування

Для функціонування Ansible потрібен тільки python. Для Ansible Master він повинен бути не нижче версії 2.6, а для Ansible Host не нижче версії 2.4, але зазвичай python входить до складу більшості дистрибутивів. Пакет Ansible входить до стандартного репозиторію Debian.

Далі виконаємо наступні команди:

echo deb http://ppa.launchpad.net/ansible/ansible/ubuntu focal main >> /etc/apt/source.list

apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 93C4A3FD7BB9C367

apt update && apt install –y ansible

cd /etc/ansible/

touch ansible.cfg

touch hosts

Тепер потрібно відредагувати файл конфігурації:

nano ansible.cfg

Файл конфігурації повинен виглядати наступним чином:

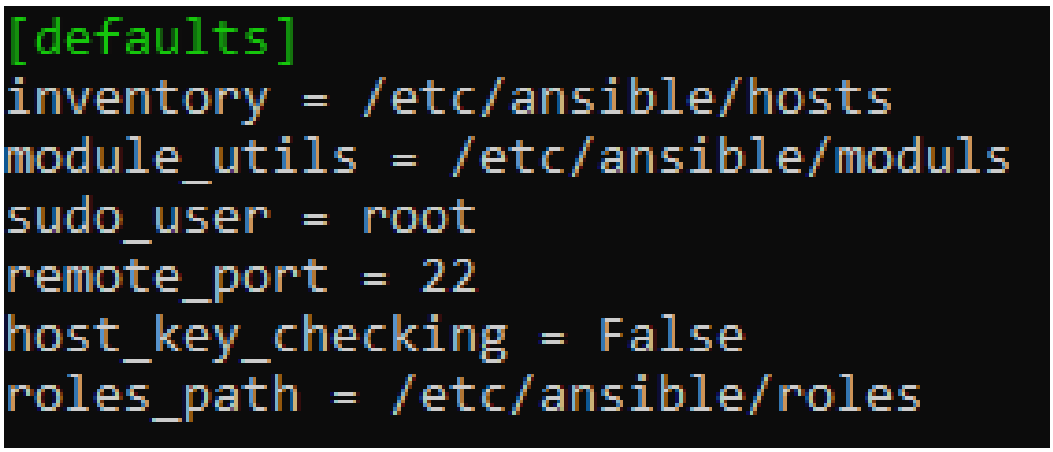

Рисунок 4.1 – Вигляд файлу конфігурації ansible

Тепер потрібно відредагувати файл інвентару.

Він повинен виглядати наступним чином:

```
[example]
host1 ansible_host=10.1.1.253
[example:vars]
ansible_user=root
ansible_password=1234
```

Рисунок 4.2 – Вигляд файлу hosts

Цього буде достатньо для демонстрації роботи.

4.2 Приклади роботи

Приклад 1. Модуль ping.

Запит ping перевіряється на віддаленому хості. Цей модуль спеціально перевіряє:

– чи увімкнений віддалений хост і чи доступний він;

– чи може середовище python успішно запускати потрібні сценарії і чи взагалі встановлений пакет та бібліотеки python;

Після надсилання запиту ping на віддалений хост модуль повертає значення, що вказує, чи успішний був ping. За замовчуванням модуль ping повертає рядок «pong» у разі успіху.

Виконаємо команду:

ansible host1 -m ping, де

ansible внутрішня утиліта, яка встановлюється разом з пакетом Ansible,

host1 назва хоста відповідно по файлу hosts,

-m вказує, що ми хочемо використати модуль, а ping це назва самого модулю.

```
root@debian-master:/etc/ansible# ansible host1 -m ping
[DEPRECATION WARNING]: Distribution debian 11.2 on host host1 should use /usr/bin/python3, but is using /usr/bin/python
 for backward compatibility with prior Ansible releases. A future Ansible release will default to using the discovered
platform python for this host. See
https://docs.ansible.com/ansible/2.10/reference_appendices/interpreter_discovery.html for more information. This
feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
host1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
root@debian-master:/etc/ansible#
```

Рисунок 4.3 – Вдале виконання запиту ping

Приклад 2. Модуль shell.

Модуль shell призначений для передачі команд та їх виконання в командному інтерпретаторі shell проти цільових хостів на базі Unix. На відміну від модулю command, shell працює на пряму з командним інтерпретатором, а command використовує свій, що в свою чергу не дозволяє використовувати стандартні системні змінні.

Виконаємо команду:

ansible host1 –m shell –a “cat /etc/hostname”, де

-a це ключ необхідний для додання аргументів до модулю,

“cat /etc/hostname” – аргумент.

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "cat /etc/hostname"
[DEPRECATION WARNING]: Distribution debian 11.2 on host host1 should use /usr/bin/python3, but is using /usr/bin/python
 for backward compatibility with prior Ansible releases. A future Ansible release will default to using the discovered
platform python for this host. See
https://docs.ansible.com/ansible/2.10/reference_appendices/interpreter_discovery.html for more information. This
feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
host1 | CHANGED | rc=0 >>
debian-host
root@debian-master:/etc/ansible#
```

Рисунок 4.4 – Приклад роботи модулю shell з утилітою cat

З модулем shell можуть працювати будь-які утиліти, які є на керованому хості.

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "dpkg -l | grep host"
[DEPRECATION WARNING]: Distribution debian 11.2 on host host1 should use /usr/bin/python3, but is using /usr/bin/python
 for backward compatibility with prior Ansible releases. A future Ansible release will default to using the discovered
platform python for this host. See
https://docs.ansible.com/ansible/2.10/reference_appendices/interpreter_discovery.html for more information. This
feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
host1 | CHANGED | rc=0 >>
ii  hostname                3.23                    amd64        utility to set/show the host name or domain na
me
ii  iputils-ping            3:20210202-1            amd64        Tools to test the reachability of network host
s
root@debian-master:/etc/ansible#
```

Рисунок 4.5 – Приклад роботи модулю shell з утилітою dpkg

Приклад 3. Модуль apt.

Модуль apt використовується для керування пакетами за допомогою системного менеджера пакетів, які є стандартними у дистрибутивах Linux. Більшість систем вимагають прав суперкористувача для керування пакетами.

На керованому хості не встановлена утиліта traceroute. Перевіримо це виконавши команду:

ansible host1 –m shell –a “dpkg –l | fgrep traceroute”

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "dpkg -l | fgrep traceroute"
[DEPRECATION WARNING]: Distribution debian 11.2 on host host1 should use /usr/bin/python3, but is using /usr/bin/python
 for backward compatibility with prior Ansible releases. A future Ansible release will default to using the discovered
platform python for this host. See
https://docs.ansible.com/ansible/2.10/reference_appendices/interpreter_discovery.html for more information. This
feature will be removed in version 2.12. Deprecation warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
host1 | FAILED | rc=1 >>
non-zero return code
root@debian-master:/etc/ansible#
```

Рисунок 4.6 – ansible, через утиліту dpkg не знайшов пакет

Встановимо пакет traceroute наступною командою:

ansible host1 –m apt –a “name=traceroute state=latest” –b, де

name – назва необхідного пакету,

state – необхідний кінцевий стан пакету на хості,

-b (become) виконує команду від суперкористувача.

Рисунок 4.7 – Вдале виконання модулю apt

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "dpkg -l | fgrep traceroute"
host1 | CHANGED | rc=0 >>
ii  traceroute                 1:2.1.0-2+b1                amd64        Traces the route taken by packets over an IPv4/IPv6 network
root@debian-master:/etc/ansible#
```

Рисунок 4.8 – ansible, через утиліту dpkg знайшов пакет

Приклад 4. Модуль iptables.

Модуль iptables використовується для налаштування, підтримки та перевірки таблиць фільтрації IP-пакетів в однойменному фільтрі. Має такий самий функціонал, як і в утиліти iptables в ОС Linux. Для змін потрібні права суперкористувача.

Зараз на debian-host пуста таблиця фільтрації. Перевірим це наступною командою:

ansible host1 –m shell –a “iptables -S”

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "iptables -S"
host1 | CHANGED | rc=0 >>
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
root@debian-master:/etc/ansible#
```

Рисунок 4.9 – Пуста таблиця iptables

Додаймо туди правило командою:

ansible –m iptables –a “chain=INPUT jump=ACCEPT source=1.2.3.4”, де

chain – цепочка в яку потрібно додати правило,

jump – дія з пакетом, яку потрібно зробити,

source – джерело звідки повинен прийти пакет.

```
root@debian-master:/etc/ansible# ansible host1 -m iptables -a "chain=INPUT jump=ACCEPT source=1.2.3.4"
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "chain": "INPUT",
    "changed": true,
    "flush": false,
    "ip_version": "ipv4",
    "rule": "-s 1.2.3.4 -j ACCEPT",
    "state": "present",
    "table": "filter"
}
root@debian-master:/etc/ansible# ansible host1 -m shell -a "iptables -S"
host1 | CHANGED | rc=0 >>
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -s 1.2.3.4/32 -j ACCEPT
root@debian-master:/etc/ansible#
```

Рисунок 4.10 – Таблиця iptables з новим правилом

Завдяки модулю iptables є можливість повноцінно працювати з таблицями правил iptables віддалено. Модуль працює, як проксі.

Приклад з таблицею nat і форфардінгом портів:

ansible host1 -m iptables -a "table=nat chain=PREROUTING in_interface=enp0s3 jump=REDIRECT protocol=tcp match=tcp destination_port=1234 to_ports=4321"

```
root@debian-master:~# ansible host1 -m iptables -a "table=nat chain=PREROUTING in_interface=enp0s3 jump=REDIRECT protocol=tcp match=tcp d
estination_port=1234 to_ports=4321"
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "chain": "PREROUTING",
    "changed": true,
    "flush": false,
    "ip_version": "ipv4",
    "rule": "-p tcp -m tcp -j REDIRECT -i enp0s3 --destination-port 1234 --to-ports 4321",
    "state": "present",
    "table": "nat"
}
root@debian-master:~# ansible host1 -m shell -a "iptables -S -t nat"
host1 | CHANGED | rc=0 >>
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-A PREROUTING -i enp0s3 -p tcp -m tcp --dport 1234 -j REDIRECT --to-ports 4321
root@debian-master:~#
```

Рисунок 4.11 – Таблиця nat в iptables з новим правилом

Приклад 5. Модуль service.

Модуль service використовується для взаємодії із системним менеджером служб в ОС Linux. Працює як проксі, просто, передає команди в системний менеджер.

Приклад приведу на демоні cron. Виконаємо наступно команду, що дізнатися, який зараз статус у нього:

ansible host1 –m shell –a “systemctl status cron”

```
root@debian-master:~# ansible host1 -m shell -a "systemctl status cron"
host1 | CHANGED | rc=0 >>
● cron.service - Regular background program processing daemon
     Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-03-21 07:45:45 CDT; 1 day 9h ago
       Docs: man:cron(8)
   Main PID: 400 (cron)
      Tasks: 1 (limit: 1133)
     Memory: 1.5M
        CPU: 521ms
     CGroup: /system.slice/cron.service
             └─400 /usr/sbin/cron -f

Mar 22 14:17:01 debian-host CRON[10072]: pam_unix(cron:session): session closed for user root
Mar 22 15:17:01 debian-host CRON[10137]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Mar 22 15:17:01 debian-host CRON[10138]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 22 15:17:01 debian-host CRON[10137]: pam_unix(cron:session): session closed for user root
Mar 22 16:17:01 debian-host CRON[10268]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Mar 22 16:17:01 debian-host CRON[10269]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 22 16:17:01 debian-host CRON[10268]: pam_unix(cron:session): session closed for user root
Mar 22 17:17:01 debian-host CRON[10669]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Mar 22 17:17:01 debian-host CRON[10670]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 22 17:17:01 debian-host CRON[10669]: pam_unix(cron:session): session closed for user root
```

Рисунок 4.12 – Статус демона cron

Як видно по виводу менеджера служб, на данний момент демон cron активний. Зупинимо його наступною командою:

ansible host1 –m service –a “name=cron state=stopped”

```
root@debian-master:~# ansible host1 -m service -a "name=cron state=stopped"
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "name": "cron",
    "state": "stopped",
    "status": {
```

Рисунок 4.13 – Вдале виконання ad hoc команди з модулем service

Перевіримо стан демона.

```
root@debian-master:~# ansible host1 -m shell -a "systemctl status cron"
host1 | FAILED | rc=3 >>
● cron.service - Regular background program processing daemon
     Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: enabled)
     Active: inactive (dead) since Tue 2022-03-22 17:26:00 CDT; 49s ago
       Docs: man:cron(8)
    Process: 400 ExecStart=/usr/sbin/cron -f $EXTRA_OPTS (code=killed, signal=TERM)
   Main PID: 400 (code=killed, signal=TERM)
        CPU: 521ms

Mar 22 15:17:01 debian-host CRON[10137]: pam_unix(cron:session): session closed for user root
Mar 22 16:17:01 debian-host CRON[10268]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Mar 22 16:17:01 debian-host CRON[10269]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 22 16:17:01 debian-host CRON[10268]: pam_unix(cron:session): session closed for user root
Mar 22 17:17:01 debian-host CRON[10669]: pam_unix(cron:session): session opened for user root(uid=0) by (uid=0)
Mar 22 17:17:01 debian-host CRON[10670]: (root) CMD (   cd / && run-parts --report /etc/cron.hourly)
Mar 22 17:17:01 debian-host CRON[10669]: pam_unix(cron:session): session closed for user root
Mar 22 17:26:00 debian-host systemd[1]: Stopping Regular background program processing daemon...
Mar 22 17:26:00 debian-host systemd[1]: cron.service: Succeeded.
Mar 22 17:26:00 debian-host systemd[1]: Stopped Regular background program processing daemon.non-zero return code
```

Рисунок 4.14 – Статус демона cron після виконання ad hoc команди модуля service

Також модуль дозволяє адмініструвати автозапуск служб.

```
root@debian-master:~# ansible host1 -m shell -a "systemctl list-unit-files | grep cron"
host1 | CHANGED | rc=0 >>
cron.service                                              enabled         enabled
root@debian-master:~# ansible host1 -m service -a "name=cron enabled=no" | head -n 10
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "enabled": false,
    "name": "cron",
    "status": {
        "ActiveEnterTimestampMonotonic": "0",
        "ActiveExitTimestampMonotonic": "0",
root@debian-master:~# ansible host1 -m shell -a "systemctl list-unit-files | grep cron"
host1 | CHANGED | rc=0 >>
cron.service                                              disabled        enabled
root@debian-master:~#
```

Рисунок 4.15 – Стан автозапуску демона cron до і після виконання ad hoc команд

Приклад 6. Модуль lineinfile.

Модуль lineinfile використовується для додавання або зміни одного рядку у файлі.

Для прикладу створимо файл на віддаленому хості.

```
root@debian-master:~# ansible host1 -m shell -a "echo Hello World > /root/test && echo Hello DDT >> /root/test && echo Hello KID-43 >> /root/test"
host1 | CHANGED | rc=0 >>

root@debian-master:~# ansible host1 -m shell -a "cat /root/test"
host1 | CHANGED | rc=0 >>
Hello World
Hello DDT
Hello KID-43
root@debian-master:~#
```

Рисунок 4.16 – Створення тестового файлу та його вміст

Відредагуємо файл командою:

ansible host1 -m lineinfile -a "path=/root/test regexp='Hello DDT' line='Hello DUT'"

```
root@debian-master:~# ansible host1 -m lineinfile -a "path=/root/test regexp='Hello DDT' line='Hello DUT'"
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "backup": "",
    "changed": true,
    "msg": "line replaced"
}
root@debian-master:~# ansible host1 -m shell -a "cat /root/test"
host1 | CHANGED | rc=0 >>
Hello World
Hello DUT
Hello KID-43
root@debian-master:~#
```

Рисунок 4.17 – Кінцевий результат зміни тестового файлу

Приклад 7. Модуль blockinfile.

Модуль blockinfile використовується для вставки, оновлення та видалення блоку рядків. Блок буде оточений спеціальним маркером, таким як, початок і кінець, щоб зробити завдання ідемпотентним.

Використаємо файл із попереднього прикладу.

Виконаємо команду:

ansible host1 -m blockinfile -a "path=/root/test block='lineinfile\n blockinfile'"

```
root@debian-master:~# ansible host1 -m shell -a "cat /root/test"
host1 | CHANGED | rc=0 >>
Hello World
Hello DUT
Hello KID-43
root@debian-master:~# ansible host1 -m blockinfile -a "path=/root/test block='lineinfile\n blockinfile'"
host1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": true,
    "msg": "Block inserted"
}
root@debian-master:~# ansible host1 -m shell -a "cat /root/test"
host1 | CHANGED | rc=0 >>
Hello World
Hello DUT
Hello KID-43
# BEGIN ANSIBLE MANAGED BLOCK
lineinfile
 blockinfile
# END ANSIBLE MANAGED BLOCK
root@debian-master:~#
```

Рисунок 4.18 – Вдале виконання модулю blockinfile

Тепер наведу приклад playbook.

Створимо по шляху /etc/ansible/ yml файл плейбуку.

Помістимо в нього наступний вміст:

```
---
 - name: rsyslog settings
   hosts: host1
   become: yes
   tasks:
      - name: install rsyslog-gnutls
        apt:
          name: rsyslog-gnutls
          state: latest
      - name: add ip gsplunk in hosts
        shell: echo 53.324.584.221 gsplunk >> /etc/hosts
      - name: add gsplunk settings in rsyslog.conf
        run_once: true
        blockinfile:
          path: /etc/rsyslog.conf
          insertafter: "WorkDirectory"
          block: |
           $DefaultNetstreamDriverCAFile /rootCA.crt
           $DefaultNetstreamDriver gtls # use gtls netstream driver
           $ActionSendStreamDriverMode 1 # require TLS for the connection
           $ActionSendStreamDriverAuthMode anon # server is NOT authenticated
           *.* @@gsplunk:515
          backup: yes
      - name: restart rsyslog
        service:
            name: rsyslog
            state: restarted
...
```

Рисунок 4.19 – Вигляд файлу сценарію

Запустимо виконання команд із плейбуку:

ansible-playbook example.yml

```
root@debian-master:/etc/ansible# ansible host1 -m shell -a "dpkg -l | grep rsyslog-gnutls"
host1 | CHANGED | rc=0 >>
ii  rsyslog-gnutls               8.2102.0-2                  amd64        TLS protocol support for rsyslog (GnuTLS)
root@debian-master:/etc/ansible# ansible host1 -m shell -a "cat /etc/hosts"
host1 | CHANGED | rc=0 >>
127.0.0.1       localhost
127.0.1.1       debian-master

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
53.324.584.221 gsplunk
root@debian-master:/etc/ansible# ansible host1 -m shell -a "cat /etc/rsyslog.conf | grep Driver"
host1 | CHANGED | rc=0 >>
$DefaultNetstreamDriverCAFile /rootCA.crt
$DefaultNetstreamDriver gtls # use gtls netstream driver
$ActionSendStreamDriverMode 1 # require TLS for the connection
$ActionSendStreamDriverAuthMode anon # server is NOT authenticated
root@debian-master:/etc/ansible#
```

Рисунок 4.20 – Результати виконання завдань із playbook

4.3 Адміністрування серверу Wireguard за допомогою Ansible

Далі ми перейдемо до наведення прикладу уже готових до використання плейбуків для Ansible по сетапу Wireguard серверу та їх реального застосування на сервері з операційною системою Debian 11 та публічним IP-адресом. Цей приклад допоможе вам краще зрозуміти, як Ansible дозволяє автоматизувати даний процес.

Перший та основний плейбук займається встановленням необхідних пакетів для роботи серверу Wireguard та створює файли конфігурації сервера і першого клієнта. Також він додає параметри до файлу sysctl.conf, які відключають IPv6 на сервері та дозволяють форвардинг IPv4 пакетів. Частина коду приведена на рисунку 4.21

Рисунок 4.21 – Playbook для встановлення Wireguard серверу

Другий, вже допоміжний плейбук займається додаванням в кінець файлу конфігурації серверу нових клієнтів (пірів) та генеруванням їх файлів конфігурації для підключення до серверу. Весь його код приведено на наступному рисунку 4.22

Рисунок 4.22 – Playbook для додаванная нового peer у файл конфігурації Wireguard

Третій, і вже останній допоміжний плейбук дозволяє змінювати ключі впн клієнтів, що в свою чергу допомагає їх “заблокувати”. Код доступний на рисунку 4.23

```
---
- name: set up wireguard server
  hosts: 127.0.0.1
  become: yes
  vars:
    IP: 10.66.66.4/32
    client: client3
    wgport: 52262
    wireguard_dir: /etc/wireguard
    wireguard_conf: "{{ wireguard_dir }}/server.conf"
    serverpk: "{{ wireguard_dir }}/keys/server-private.key"
    serverpubk:  "{{ wireguard_dir }}/keys/server-public.key"
    serverpsk: "{{ wireguard_dir }}/keys/serverpsk"
    clientpk: "{{ wireguard_dir }}/keys/client-private.key"
    clientpubk:  "{{ wireguard_dir }}/keys/client-public.key"
  tasks:
    - run_once: true
      shell:
        touch "{{ wireguard_conf }}" ;
        mkdir -p "{{ wireguard_dir }}"/keys

    - shell:
        wg genkey | tee "{{ clientpk }}" | wg pubkey > "{{ clientpubk }}";
        wg genpsk > "{{ serverpsk }}"

    - shell: cat "{{ serverpsk }}"
      register: rserverpsk

    - shell: cat "{{ serverpk }}"
      register: rserverpk

    - shell: cat "{{ serverpubk }}"
      register: rserverpubk

    - shell: cat "{{ clientpubk }}"
      register: rclientpubk

    - shell: cat "{{ clientpk }}"
      register: rclientpk

    - name: change server config
      replace:
        path: "{{ wireguard_conf }}"
        regexp: '^#{{ client }}[\n\w\W]*^AllowedIPs = {{ IP }}$'
        replace: "#{{ client }}\nPublicKey = {{ rclientpubk.stdout }}\n"
                 "#PrivateKey = {{ rclientpk.stdout }}\nPresharedKey = {{ rserverpsk.stdout }}\nAllowedIPs = {{ IP }}"
      notify: restart wireguard

    - name: create client config file
      blockinfile:
        path: "{{ wireguard_dir }}/{{ client }}.conf"
        marker: ""
        create: true
        block: |
          [Interface]
          PrivateKey = {{ rclientpk.stdout }}
          Address = {{ IP }}
          DNS = 1.1.1.1, 8.8.8.8
          [Peer]
          PresharedKey = {{ rserverpsk.stdout }}
          Publickey = {{ rserverpubk.stdout }}
          AllowedIPs = 10.66.66.1/32, 0.0.0.0/1, 128.0.0.0/1
          PersistentKeepalive = 30
          Endpoint = {{ ansible_default_ipv4.address }}:{{ wgport }}

    - lineinfile:
        path: "{{ wireguard_dir }}/{{ client }}.conf"
        regex: "^$"
        state: absent

    - shell: cat "{{ wireguard_conf }}"
      register: serverresult

    - shell: cat "{{ wireguard_dir }}/{{ client }}.conf"
      register: clientresult

    - debug:
        var: serverresult.stdout_lines

    - debug:
        var: clientresult.stdout_lines

  handlers:
    - name: restart wireguard
      service:
        name: wg-quick@server
        state: restarted
...
```

Рисунок 4.23 – Playbook для зміну ключів peer

Запустимо плейбук для сетапу wireguard серверу. Після виконання основної частини коду, для зручності, буде виведено зміст файлу конфігурації сервера та файлу конфігурації клієнта.

```
TASK [debug] *******************************************************************************************************************
ok: [diplom] => {
    "serverresult.stdout_lines": [
        "[Interface]",
        "Address = 10.66.96.1/24",
        "ListenPort = 52263",
        "#Publickey = 3xZutX8ltNzIH3zVoJzBU3L6drCW70uGUcEJwfxfGXA=",
        "PrivateKey = qLnbWziPT8kU2AfS6yhWWhcbs9tRdk2p1N80hBQH4lY=",
        "PostUp = iptables -A INPUT -p udp -m udp --dport 52263 -j ACCEPT;",
        "PostUp = iptables -A FORWARD -i %i -j ACCEPT;",
        "PostDown = iptables -D FORWARD -i %i -j ACCEPT;",
        "PostDown = iptables -D INPUT -p udp -m udp --dport 52263 -j ACCEPT;",
        "[Peer]",
        "Publickey = b+hORSxiky2dz1X9TuUk6BmQUDorvX3YHNNWeqk6rWY=",
        "#PrivateKey = GIenW+/IWl3lN18hBNE3CNQ9VT1Okb6NcbIuL9s2NXo=",
        "PresharedKey = 19O+9PhDi1BC9j0E/iuPQjuM5tGWNPFhTrtqc7aM/Hk=",
        "AllowedIPs = 10.66.96.2/32"
    ]
}

TASK [debug] *******************************************************************************************************************
ok: [diplom] => {
    "clientresult.stdout_lines": [
        "[Interface]",
        "PrivateKey = GIenW+/IWl3lN18hBNE3CNQ9VT1Okb6NcbIuL9s2NXo=",
        "Address = 10.66.96.2/24",
        "DNS = 1.1.1.1, 8.8.8.8",
        "[Peer]",
        "PresharedKey = 19O+9PhDi1BC9j0E/iuPQjuM5tGWNPFhTrtqc7aM/Hk=",
        "Publickey = 3xZutX8ltNzIH3zVoJzBU3L6drCW70uGUcEJwfxfGXA=",
        "AllowedIPs = 10.66.96.1/32, 0.0.0.0/1, 128.0.0.0/1",
        "PersistentKeepalive = 30",
        "Endpoint = 130.0.239.226:52263"
    ]
}

PLAY RECAP *********************************************************************************************************************
diplom                     : ok=21   changed=15   unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

Рисунок 4.24 – Приклад використання модулю debug у Ansible

Далі давайте створимо ще один пір за допомогою допоміжного плейбука. Він також виводить в кінці актуальній зміст файлів конфігурації.

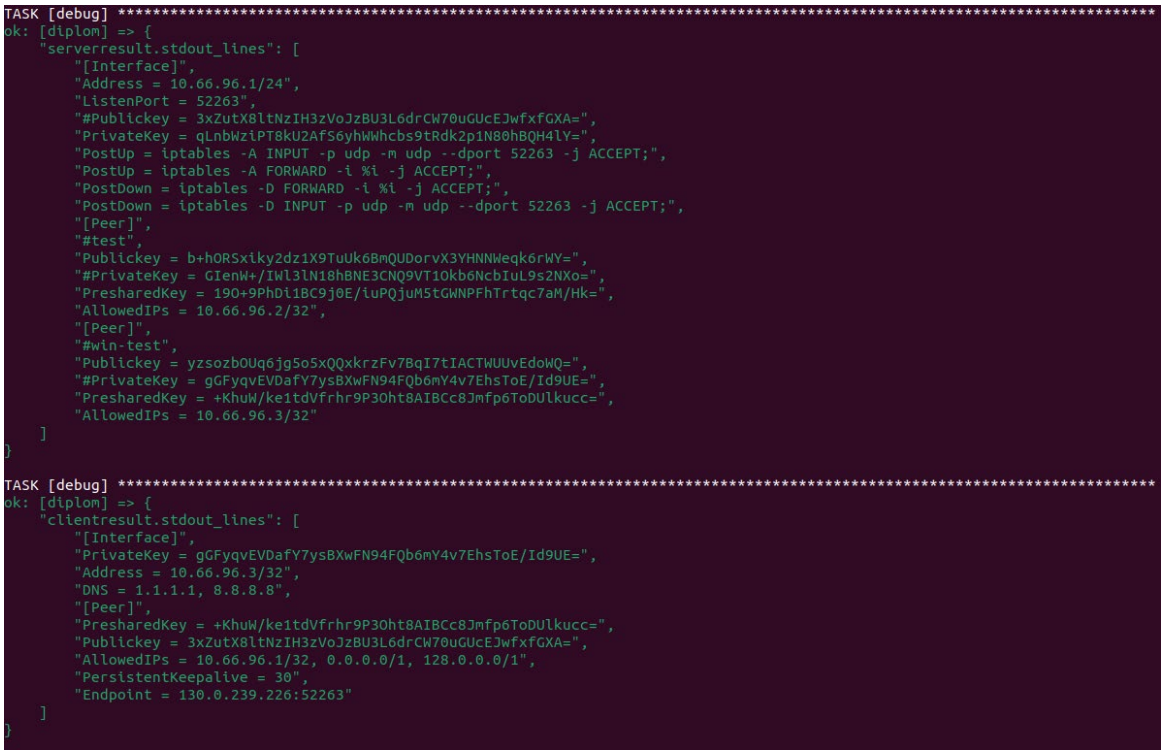

Рисунок 4.25 – Приклад файлу конфігурації сервера з новим peer

Перевіримо працездатність сервера та плейбуків за допомогою клієнта Wireguard на Windows, який доступний на офіційному сайті Wireguard. Але перед цим включити саму службу wireguard серверу із використанням потрібного нам файла конфігурації.

Зробити це можна наступною командою: systemctl start wg-quick@server-test

Якщо служба успішно була включена то помилок при виконанні попередньої команди не повинно було виникнути. Одним із способів перевірки чи працює сервер Wireguard є використання утиліти яка входить в стандартний пакет Wireguard сервера: wg.

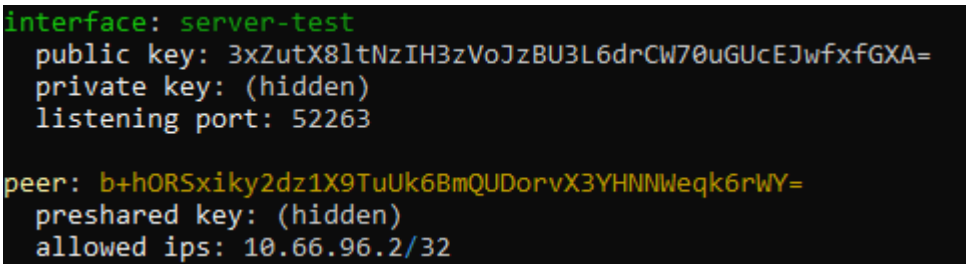

Рисунок 4.26 – Вивід утиліти wg

Після встановлення клієнту Wireguard імпортуємо файл конфігурації клієнта, який генерується плейбуком та спробуємо підключитись.

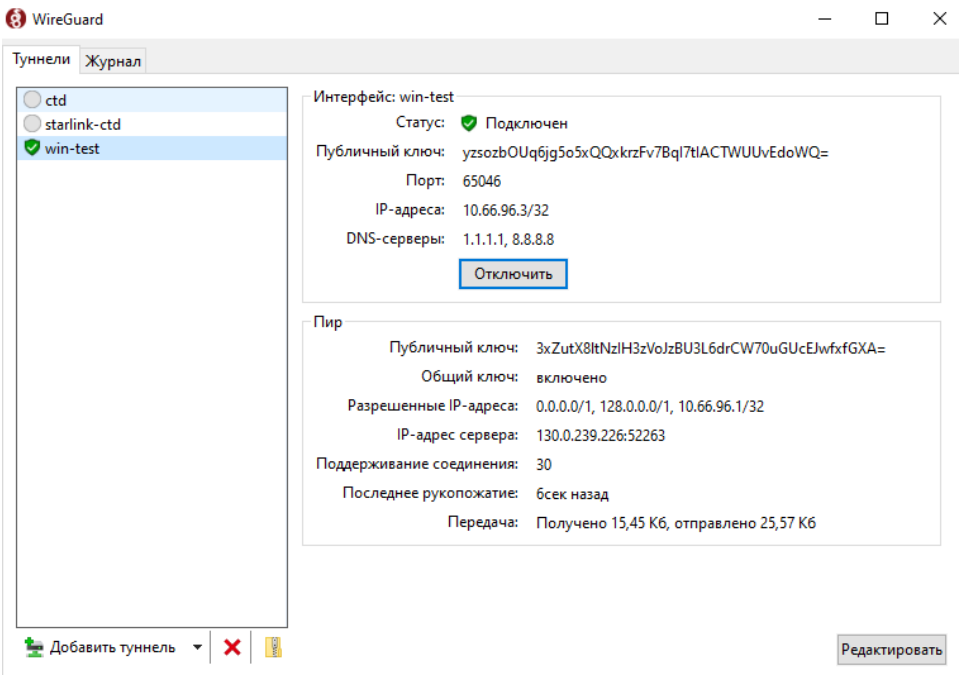

Рисунок 4.27 – Вікно клієнта Wireguard з успішним рукостисканням

Як можна побачити на скріншоті у нас успішне рукостискання це в свою чергу означає, що клієнт успішно приєднався до серверу. Давайте перевіримо таблицю маршрутизації, що дізнатись які ми отримали маршрути від впн серверу.

Зробити це на Windows можна виконавши команду в cmd: route print

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0      192.168.88.1   192.168.88.250   4250
          0.0.0.0          0.0.0.0         On-link      172.16.34.12     26
          0.0.0.0        128.0.0.0         On-link        10.66.96.3      5
       5.45.79.30  255.255.255.255      192.168.88.1   192.168.88.250   4251
       10.66.96.1  255.255.255.255         On-link        10.66.96.3      5
       10.66.96.3  255.255.255.255         On-link        10.66.96.3    261
        127.0.0.0        255.0.0.0         On-link         127.0.0.1   4556
        127.0.0.1  255.255.255.255         On-link         127.0.0.1   4556
  127.255.255.255  255.255.255.255         On-link         127.0.0.1   4556
  127.255.255.255  255.255.255.255         On-link        10.66.96.3    261
        128.0.0.0        128.0.0.0         On-link        10.66.96.3      5
     172.16.34.12  255.255.255.255         On-link      172.16.34.12    281
     192.168.56.0    255.255.255.0         On-link      192.168.56.1   4506
     192.168.56.1  255.255.255.255         On-link      192.168.56.1   4506
   192.168.56.255  255.255.255.255         On-link      192.168.56.1   4506
     192.168.88.0    255.255.255.0         On-link    192.168.88.250   4506
   192.168.88.250  255.255.255.255         On-link    192.168.88.250   4506
   192.168.88.255  255.255.255.255         On-link    192.168.88.250   4506
        224.0.0.0        240.0.0.0         On-link         127.0.0.1   4556
        224.0.0.0        240.0.0.0         On-link      192.168.56.1   4506
        224.0.0.0        240.0.0.0         On-link    192.168.88.250   4506
        224.0.0.0        240.0.0.0         On-link      172.16.34.12     26
  255.255.255.255  255.255.255.255         On-link         127.0.0.1   4556
  255.255.255.255  255.255.255.255         On-link      192.168.56.1   4506
  255.255.255.255  255.255.255.255         On-link    192.168.88.250   4506
```

Рисунок 4.28 – Актуальні маршрути з виводу команди route print

Керуючись даними які ми отримали зі скріншота можна зробити вивід, що весь трафік зараз повинен йти через впн сервер. Давайте перевіримо.

Для цього виконаємо команду в cmd: tracert -d 1.1.1.1

```
C:\Users\admin>tracert -d 1.1.1.1

Tracing route to 1.1.1.1 over a maximum of 30 hops

  1   129 ms   127 ms   129 ms  10.66.96.1
  2   125 ms   137 ms   120 ms  130.0.237.90
  3   125 ms   118 ms   117 ms  130.0.232.1
  4   125 ms   138 ms   134 ms  62.205.159.160
  5   114 ms   114 ms   113 ms  62.205.132.7
  6     *        *        *     Request timed out.
  7   127 ms   127 ms   135 ms  1.1.1.1

Trace complete.
```

Рисунок 4.29 – Вивід утиліти tracert

Як можна помітити із виводу утиліти tracert весь наш трафік йде через шлюз впн серверу Wireguard. Керуючись цим можна зробити вивід, що для доступу до інтернету ми користуємось публічним IP-адресом сервера на якому встановлений Wireguard.

Перевіримо це за допомогою команди в cmd: curl ifconfig.co

```
C:\Users\admin>curl ifconfig.co
130.0.239.226
```

Рисунок 4.30 – Вивід утиліти curl

Отримавши вивід утиліти можна твердо бути впевними, що весь трафік на стороні клієнта Wireguard проходить через шлюз впн серверу, що в свою чергу дозволяє стверджувати, що трафік використовує публічний IP-адрес сервера для доступу до інтернету.

Тепер змінимо ключі для піра win-test за допомогою плейбуку.

```
TASK [debug] *******************************************************************************************************************************
ok: [diplom] => {
    "serverresult.stdout_lines": [
        "[Interface]",
        "Address = 10.66.96.1/24",
        "ListenPort = 52263",
        "#Publickey = 3xZutX8ltNzIH3zVoJzBU3L6drCW70uGUcEJwfxfGXA=",
        "PrivateKey = qLnbWzlPT8kU2AfS6yhWWhcbs9tRdk2p1N80hBQH4lY=",
        "PostUp = iptables -A INPUT -p udp -m udp --dport 52263 -j ACCEPT;",
        "PostUp = iptables -A FORWARD -i %i -j ACCEPT;",
        "PostDown = iptables -D FORWARD -i %i -j ACCEPT;",
        "PostDown = iptables -D INPUT -p udp -m udp --dport 52263 -j ACCEPT;",
        "[Peer]",
        "#test",
        "Publickey = b+hORSxiky2dz1X9TuUk6BmQUDorvX3YHNNWeqk6rWY=",
        "#PrivateKey = GIenW+/IWl3lN18hBNE3CNQ9VT1Okb6NcbIuL9s2NXo=",
        "PresharedKey = 19O+9PhDi1BC9j0E/iuPQjuM5tGWNPFhTrtqc7aM/Hk=",
        "AllowedIPs = 10.66.96.2/32",
        "[Peer]",
        "#win-test",
        "PublicKey = uXdSm7/NyrmZSBkNS6NkU2t0qeZb5wA2176zYmZdniA=",
        "#PrivateKey = mFjs2wNs8N4uxIDb7FfxocrCGu2WBgOs2lfqWxIPtX8=",
        "PresharedKey = GtNupgAUy741T+n20xlyPVxca4bRTEq9BndSRBg8klg=",
        "AllowedIPs = 10.66.96.3/32"
    ]
}
```

Рисунок 4.31– Файл конфігурації зі зміненими ключами для peer win-test

Якщо зрівняти попередні скріншоти з виводом файлу конфігурації серверу то є можливість помітити, що у піра win-test змінились його ключі. Із цього робимо вивід, що в нього зник доступ до впн серверу. Давайте перевіримо, для цього спробуємо знову підключитись до впн серверу за допомогою клієнта Wireguard.

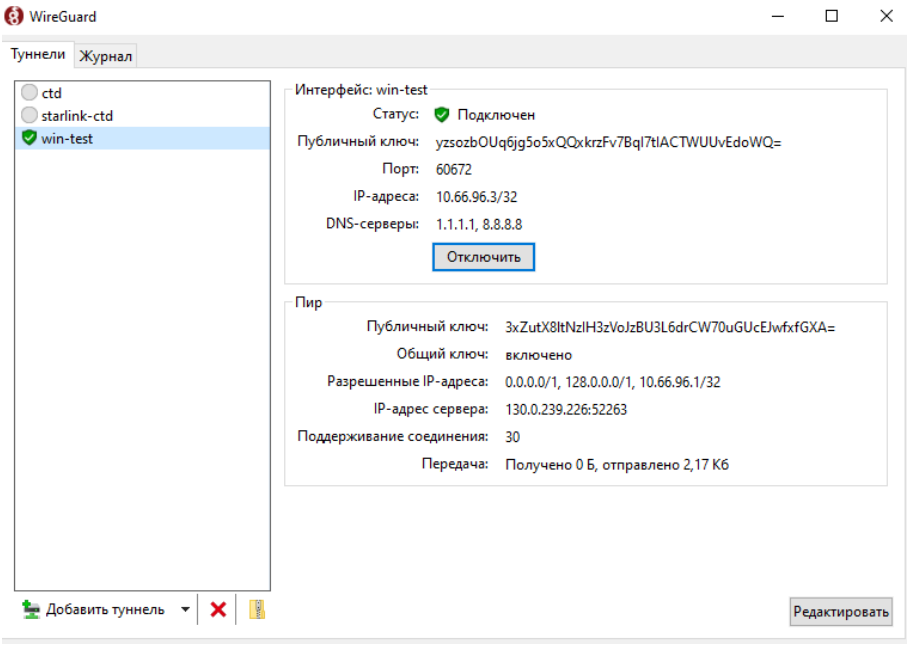

Рисунок 4.32 – Вікно клієнта Wireguard з невдалим рукостисканням

Як видно із скріншоту у нас зник рядок рукостискання. Це може означати, що впн сервер зараз не доступний, або ми не можемо до нього підключитись. Давайте переглянемо логи клієнта Wireguard.

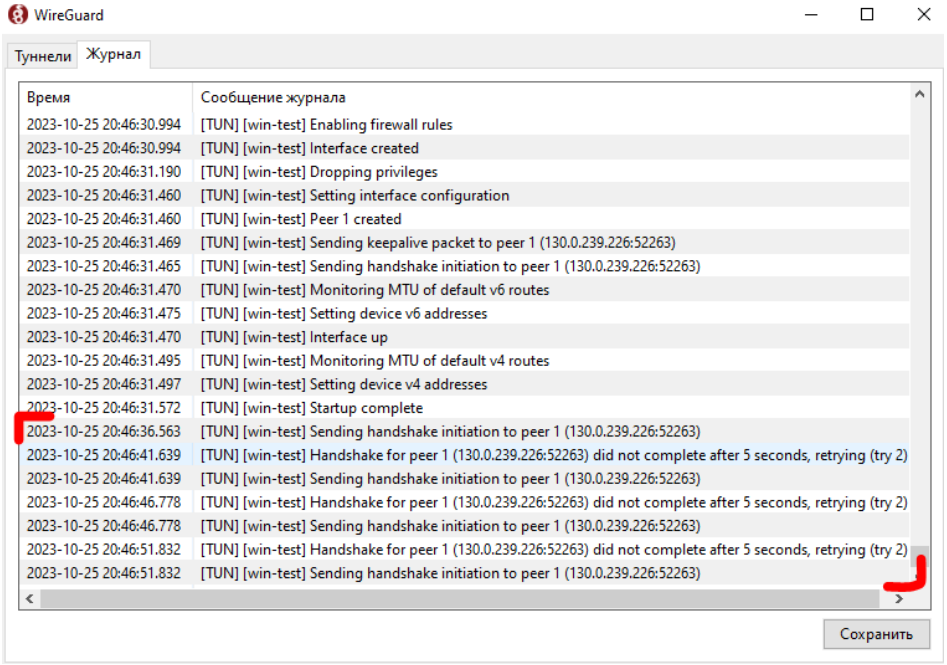

Рисунок 4.33 – Вікно клієнта Wireguard з логами

Як видно з логу клієнта Wireguard у нас дійсно зник доступ до серверу, а саме можливість підключення до впн за допомогою наявних ключів.

Спробуємо підключитись за допомогою конфігурації з новими ключами.

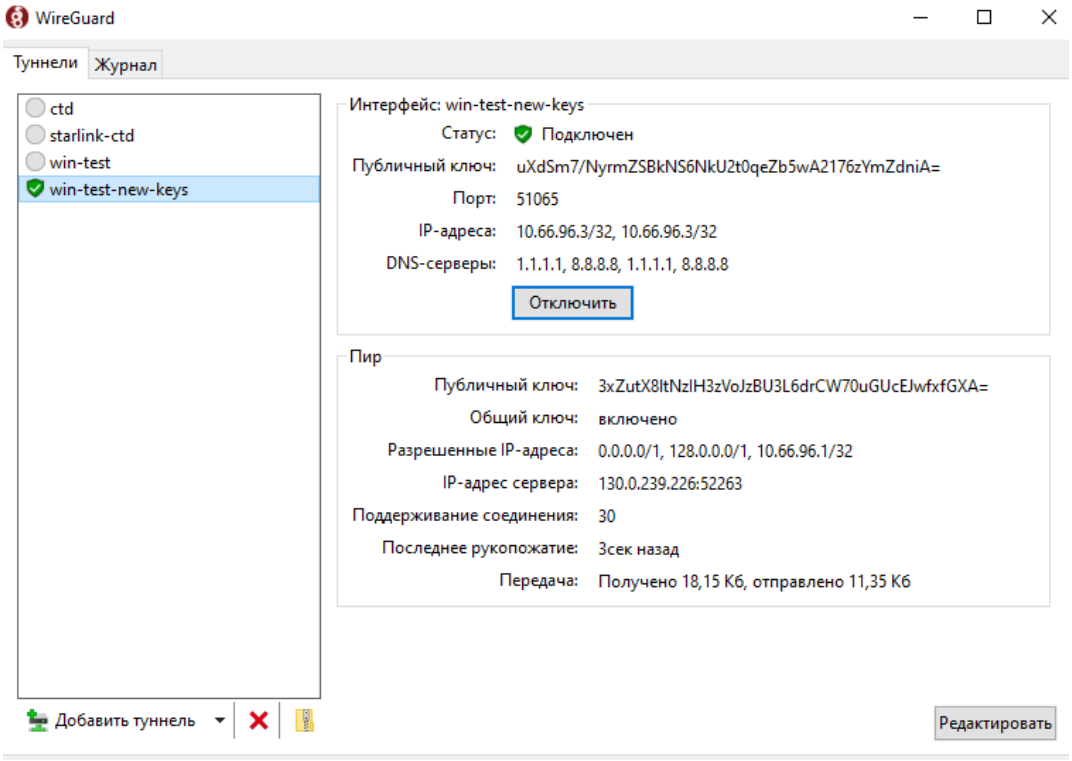

Рисунок 4.34 – Вікно клієнта Wireguard з успішним рукостисканням на нових ключах

Як видно зі скріншоту у нас знову успішне рукостискання. Давайте перевіримо.

```
C:\Users\admin>tracert -d 1.1.1.1

Tracing route to 1.1.1.1 over a maximum of 30 hops

  1    123 ms    137 ms    120 ms  10.66.96.1
  2    132 ms    219 ms    122 ms  130.0.237.90
  3    196 ms    120 ms    150 ms  130.0.232.1
  4    128 ms    127 ms    132 ms  62.205.159.160
  5    129 ms    134 ms    142 ms  62.205.132.7
  6     *         *         *      Request timed out.
  7    142 ms    132 ms    132 ms  1.1.1.1

Trace complete.

C:\Users\admin>curl ifconfig.co
130.0.239.226
```

Рисунок 4.35 – Вивід утиліт tracert та curl

Зробивши аналіз виводу утиліт з рисунку 4.35 можна зробити вивід, що рукостискання дісно успішне і впн сервер являється шлюзом за замовчуванням для клієнта.

Зі сторони серверу успішне рукостискання віглядає ось так:

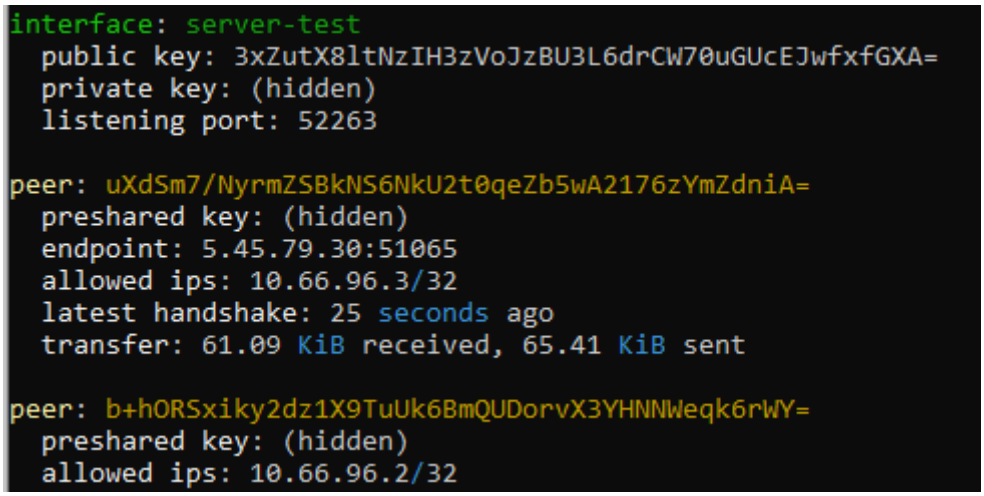

Рисунок 4.36 – Приклад успішного рукостискання зі сторони серверу

Як можна побачити по скрінштах доступ до впн серверу відновися як тільки ми завантажили файл конфігурації з новими ключами.

ВИСНОВОК

Сучасна операційна система являє собою дуже складний комплекс програмних засобів, що надають користувачам доступ до інформації і управління процесами в самому комп'ютері, а це, в свою чергу, дозволяє спростити роботу з ним. Програмний інтерфейс операційних систем дозволяє зменшити кінцевий розмір конкретних програм за рахунок уніфікації системних процесів та спрощує роботу з усіма компонентами обчислювальної технікі.

Кожна операційна система, визначає набір функцій, що забезпечують обмін інформацією між процесами, службами та файлами, які, в свою чергу, складаються із відкриття, читання, управління та закриття файлу.

Версія операційної системи на платформі Unix – Debian, на мою думку, являється найбільш сучасною та зручною із сімейства Linux за рахунок постійних оновлень і величезного обсягу доступного програмного забезпечення в репозиторіях. За рахунок цього вона є найпопулярнішою в серверному сегменті.

Сучасній системи управління конфігурацією представляють собою програми та програмні комплекси, які дозволяють централізовано керувати великим парком комп'ютерних машин на різноманітних операційних системах. Вони в повній мірі намагаються реалізувати принцип "Інфраструктура як код". Одним із представників сімейства систем управління конфігураціями являється Ansible.

Ansible представляє із себе мінімалістичний та простий у вивченні інструмент для автоматизації рутинних процесів. Він має величезну кількість вбудованих модулів, які можна використовувати для вирішення будь-яких задач, таких як встановлення пакетів чи зміна значення у файлі конфігурації. Його спрощені вимоги до інфраструктури та доступний синтаксис підходять тим, хто починає працювати із системами управління конфігураціями.

Технологія Wireguard є досить простою в конфігурації, ефективною та безпечною з точки зору забезпечення приватності та безпеки даних при дистанційному підключенні до серверів. Її швидкість та низький рівень системних ресурсів роблять її ідеальним вибором для дистанційного доступу.

В ході роботи, було встановлено, що будь-яка операційна система створена для зручного користування та керування комп’ютерними машинами. З керуванням процесами пов’язані функції операційної системи, такими як управління використанням часу центрального процесора, файлом підкачки, буфером введення та інше.

Системи управління конфігураціями являються зручним способом адміністрування локальних мереж і не тільки їх. За допомогою систем управління конфігураціями є можливість зменшити кількість часу, який потрібно витрачати на адміністрування операційних систем завдяки вбудованих в них модулів.

ПЕРЕЛІК ПОСИЛАНЬ

1. OccupyTheWeb Linux Basics for Hackers: Getting Started with Networking, Scripting, and Security in Kali – NoStarch, 2018. – 248 с.
2. Dan Mackin, Ben Whaley, Trent R. Hein, Garth Snyder, Evi Nemeth UNIX and Linux System Administration Handbook 5th Edition — Addison-Wesley Professional, 2017. — 1232 с.
3. Christopher Negus Linux Bible 8th Edition– John Wiley & Sons, Inc., 2018. – 783 с.
4. Ansible Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.ansible.com/
5. Demystifying Ansible: Fundamental Concepts and Practical Tutorials [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/cloud-native-daily/demystifying-ansible-fundamental-concepts-and-practical-tutorials-83491b90260a
6. Ansible Playbooks [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@sayalishewale12/ansible-playbooks-0bf9336a6e19
7. Ansible Basics and its Playbook Configuration [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@deepamathan/ansible-basics-and-its-playbook-configuration-219b42915201
8. WireGuard: fast, modern, secure VPN tunnel [Електронний ресурс] – Режим доступу до ресурсу: https://www.wireguard.com/
9. TCP, TLS, IPSec and OpenSSL: The Old Dinosaurs Need Replaced? Meet WireGuard [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/asecuritysite-when-bob-met-alice/tcp-tls-ipsec-and-openssl-the-old-dinosaurs-need-replaced-meet-wireguard-eb23578563cf
10. What is WireGuard? [Електронний ресурс] – Режим доступу до ресурсу: https://cybernews.com/what-is-vpn/wireguard-protocol/
11. Debian Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.debian.org/

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)