Йона Л.Г., Педяш В.В., Мазур Г.Д.

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

Методичні рекомендації для практичних та лабораторних занять

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Міжнародний гуманітарний університет
Кафедра комп’ютерної інженерії та інноваційних технологій

Йона Л.Г., Педяш В.В., Мазур Г.Д.

БЕЗПЕКА ПРОГРАМ ТА ДАНИХ

Методичні рекомендації для практичних та лабораторних занять
здобувачів вищої освіти зі спеціальностей:
A4.09 Середня освіта (Інформатика),
F2 Інженерія програмного забезпечення,
F3 Комп'ютерні науки,
F7 Комп'ютерна інженерія,
F5 Кібербезпека та захист інформації,
G5 Електроніка, електронні комунікації, приладобудування та радіотехніка

Одеса 2025

Затверджено Вченою Радою Міжнародного гуманітарного університету (протокол № 12 від 23 червня 2025 року)

Йона Л.Г., Педяш В.В., Мазур Г.Д.

Безпека програм та даних: Методичні рекомендації для практичних та лабораторних занять [Електронне видання] / Йона Л.Г., Педяш В.В., Мазур Г.Д. — Кафедра комп'ютерної інженерії та інноваційних технологій Міжнародного гуманітарного університету. — Одеса, 2025. — 136 с.

Методичні рекомендації призначені для проведення практичних та лабораторних занять з дисципліни «Безпека програм та даних» і спрямовані на формування у здобувачів вищої освіти практичних навичок забезпечення безпеки програмного забезпечення, операційних систем і мережевих сервісів.

Посібник містить комплекс лабораторних робіт, у межах яких розглядаються питання налаштування віртуалізованого середовища для дослідження інформаційної безпеки, аудиту безпеки операційних систем, захисту від атак грубої сили, реалізації моделей контролю доступу, аналізу порушень CIA-триади, фільтрації мережевого трафіку, застосування криптографічних засобів захисту даних, організації захищених VPN-з'єднань, використання захищених протоколів передачі даних, а також моніторингу та аналізу інцидентів інформаційної безпеки.

Методичні рекомендації можуть використовуватися під час проведення практичних і лабораторних занять, а також виконання кваліфікаційних робіт. Видання адресоване студентам спеціальностей галузі інформаційних технологій, викладачам, аспірантам та фахівцям, які вивчають або викладають дисципліни, пов'язані з інформаційною та кібербезпекою.

ЗМІСТ

ВСТУП

Сучасна інженерна та наукова діяльність у галузі інформаційних технологій неможлива без ґрунтовного розуміння принципів захисту програмного забезпечення та даних. У світлі постійного зростання кількості та складності кіберзагроз, формування практичних навичок у сфері інформаційної безпеки стає ключовим завданням підготовки фахівців у галузі комп'ютерних наук, кібербезпеки та інженерії програмного забезпечення. Дисципліна «Безпека програм та даних» спрямована на ознайомлення студентів із фундаментальними концепціями, механізмами та інструментами, що забезпечують захист інформаційних систем на рівні операційної системи, мережевих протоколів, криптографічних засобів та систем моніторингу безпеки.

Збірник методичних рекомендацій призначено для використання під час проведення практичних і лабораторних занять з дисципліни «Безпека програм та даних» та охоплює 10 навчальних робіт, які послідовно розкривають основні теми курсу: від налаштування віртуалізованого середовища та аудиту безпеки операційної системи до моделювання порушень CIA-триади, захисту від атак грубої сили, налаштування моделей контролю доступу (DAC/RBAC), конфігурування мережевих фаєрволів, застосування асиметричної криптографії, організації захищених VPN-тунелів, аналізу безпечних протоколів передачі даних (SSH, TLS/SSL) та моніторингу подій безпеки в Linux-системі.

Кожна навчальна робота побудована за єдиною логічною структурою, що забезпечує системність і послідовність навчального процесу. Практичне заняття базується на розділі «Ключові положення», у якому викладено необхідні теоретичні відомості, пояснено принципи роботи безпечних механізмів, наведено приклади команд, конфігурацій та узагальнювальні висновки. Опрацювання цього матеріалу формує цілісне розуміння теми та забезпечує підготовку студентів до виконання практичних завдань.

Лабораторне заняття логічно продовжує практичну частину та виконується відповідно до розділу «Лабораторне завдання». У ньому подано структуровану покрокову інструкцію з виконання конкретних завдань — від створення віртуальних машин і налаштування мережевої взаємодії до застосування інструментів безпеки (Lynis, Fail2Ban, iptables, GnuPG, StrongSwan), аналізу журналів подій, шифрування даних, налаштування захищених протоколів та верифікації результатів. Під час виконання робіт передбачається використання сучасних інструментів безпеки в середовищі Linux (Debian 12), таких як VirtualBox, rsyslog, tcpdump, Wireshark, OpenSSH, Apache, GnuPG, StrongSwan, а також стандартних утиліт командного рядка.

Запропонована організація навчального матеріалу забезпечує логічну наступність між теоретичною та практичною складовими курсу: знання, отримані під час практичних занять, закріплюються та поглиблюються у процесі виконання лабораторних робіт. Такий підхід сприяє формуванню професійних компетентностей у сфері кібербезпеки, розвитку аналітичного

мислення, уміння самостійно виявляти вразливості, застосовувати механізми захисту та обґрунтовано оцінювати ефективність запроваджених заходів.

Набуті в результаті вивчення дисципліни компетентності створюють підґрунтя для подальшого опанування курсів, пов’язаних із мережевою безпекою, криптографією, цифровою криміналістикою, управлінням інцидентами безпеки, а також для участі у науково-дослідницькій діяльності та виконання випускних кваліфікаційних робіт у галузі кібербезпеки.

Лабораторна робота № 1

Тема: Налаштування віртуального середовища Linux для дослідження інформаційної безпеки

Мета: створення та налаштування віртуалізованого середовища на основі VirtualBox з використанням ОС Linux.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

Сучасні методи та засоби захисту програмного забезпечення та даних потребують комплексного підходу до їх вивчення, що включає як теоретичну підготовку, так і практичне застосування знань у безпечному навчальному середовищі. Віртуалізація є одним із ключових інструментів, які дозволяють створити ізольоване середовище для експериментів із різними методами захисту інформації без ризику пошкодження реальних систем чи втрати даних. Використання віртуальних машин надає студентам можливість моделювати різноманітні сценарії атак та захисту, впроваджувати та тестувати механізми безпеки, а також вивчати їх взаємодію в контексті складних інформаційних систем.

Дана лабораторна робота є першою в циклі практичних занять з дисципліни "Безпека програм та даних" і створює фундаментальну інфраструктуру для подальшого вивчення методів криптографічного захисту, аутентифікації та авторизації, технологій безпечного зберігання даних, мережевого захисту та виявлення вторгнень. Розгортання віртуалізованого середовища з використанням VirtualBox та серверних дистрибутивів Debian дозволяє створити реалістичну модель інформаційної системи, де можуть бути застосовані та перевірені різні методи та засоби захисту.

Особлива увага в цій роботі приділяється налаштуванню мережевої взаємодії між віртуальними машинами, що є критичним компонентом для подальшого вивчення мережевої безпеки, а також налагодженню SSH-з'єднань, які забезпечують захищений віддалений доступ до системи. Ці базові навички є невід'ємною частиною компетенцій фахівця з інформаційної безпеки та створюють передумови для опанування більш складних технологій захисту в майбутніх лабораторних роботах курсу.

Віртуалізоване середовище, яке студенти налаштують в ході виконання цієї лабораторної роботи, стане платформою для моделювання різноманітних сценаріїв інформаційної безпеки, включаючи захист від несанкціонованого доступу, забезпечення конфіденційності даних при передачі мережею, реалізацію механізмів контролю цілісності та впровадження систем виявлення вторгнень. Таким чином, дана робота закладає основу для комплексного практичного освоєння дисципліни "Безпека програм та даних" та формує необхідні технічні навички для подальшого професійного розвитку в галузі інформаційної безпеки.

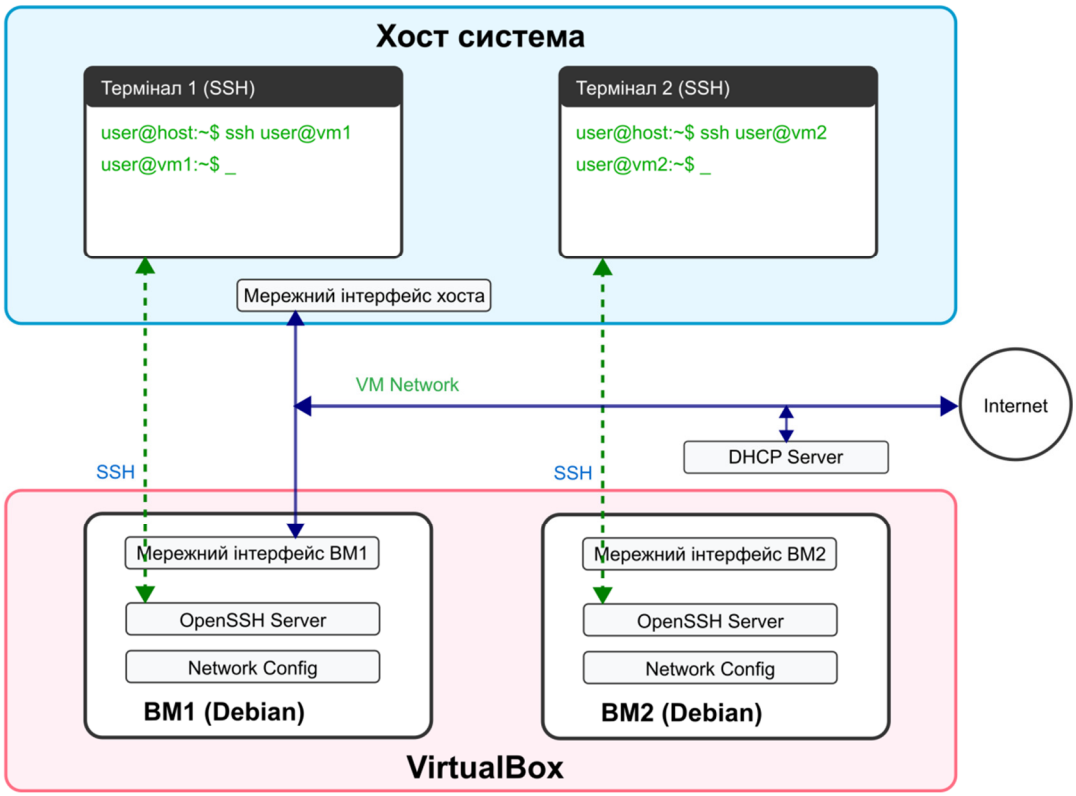

Рисунок 1.1 - Схема взаємодії компонентів віртуалізованого середовища

Представлена схема ілюструє структуру та принципи функціонування віртуалізованого середовища, побудованого на основі програмного забезпечення VirtualBox. У верхній частині зображено хост-систему з двома активними SSH-терміналами, кожен з яких забезпечує окреме захищене підключення до відповідної віртуальної машини. Хост-система виступає в ролі фізичного обчислювального ресурсу, на якому розгорнуто віртуалізоване середовище, і через власний мережевий інтерфейс забезпечує комунікацію між віртуальними машинами та доступ до зовнішніх мереж. Термінали відображають процес підключення до віртуальних машин через SSH, демонструючи команди підключення та результуючі командні рядки у середовищі віртуальних машин, що підтверджує успішне встановлення захищених з'єднань.

Мережева частина схеми представлена декількома взаємопов'язаними компонентами, що забезпечують повноцінне функціонування віртуалізованого середовища як єдиної системи. Центральним елементом мережевої інфраструктури є мережевий інтерфейс хост-системи, який взаємодіє з віртуальною мережею VM Network, що об'єднує обидві віртуальні машини. Мережева взаємодія між віртуальними машинами реалізується через створену віртуальну мережу, яка може бути налаштована в декількох режимах, включаючи NAT, Bridge, Host-only, або Internal Network, залежно від вимог до конфігурації. Доступ до зовнішніх мереж забезпечується через з'єднання VM Network з Інтернетом, при цьому адресація у віртуальній мережі може

здійснюватися за допомогою DHCP-сервера, що видає динамічні IP-адреси віртуальним машинам.

Суттєвим компонентом схеми є мережеві інтерфейси віртуальних машин, які забезпечують взаємодію віртуальних операційних систем з віртуальною мережею та, опосередковано, з хост-системою та зовнішніми мережами. Кожен мережевий інтерфейс віртуальної машини налаштований з урахуванням специфіки віртуальної мережі та має власну IP-адресу, мережеву маску та шлюз за замовчуванням, що дозволяє здійснювати маршрутизацію пакетів як всередині віртуальної мережі, так і за її межами. Налаштування мережевих інтерфейсів віртуальних машин зберігаються в конфігураційних файлах Network Config, які визначають параметри підключення до мережі, включаючи статичні або динамічні IP-адреси, DNS-сервери та інші мережеві параметри.

SSH-з'єднання між терміналами хост-системи та віртуальними машинами забезпечують захищену комунікацію, що дозволяє безпечно управляти віртуальними машинами з хост-системи без необхідності використання графічного інтерфейсу VirtualBox. Для забезпечення таких з'єднань на кожній віртуальній машині встановлено та налаштовано OpenSSH Server, який прослуховує стандартний або користувацький порт (за замовчуванням 22) та обробляє вхідні підключення, забезпечуючи автентифікацію та шифрування трафіку. Дані SSH-з'єднання проходять через віртуальну мережу, використовуючи реалізовану в VirtualBox технологію мережевої віртуалізації, що дозволяє підтримувати повноцінний мережевий стек на рівні віртуальних машин.

Доступ до Інтернету для віртуальних машин реалізується через віртуальну мережу VM Network, яка через мережевий інтерфейс хост-системи підключається до фізичної мережі та, відповідно, до глобальної мережі Інтернет. Залежно від налаштувань мережевих інтерфейсів віртуальних машин та мережевого режиму VirtualBox, доступ до Інтернету може здійснюватися через NAT (Network Address Translation), що дозволяє віртуальним машинам використовувати IP-адресу хост-системи для доступу до зовнішніх ресурсів, або через режим Bridge, який надає віртуальним машинам прямий доступ до фізичної мережі хост-системи, роблячи їх видимими як окремі мережеві пристрої.

Важливим елементом схеми є DHCP-сервер, який забезпечує автоматичне призначення IP-адрес віртуальним машинам у мережі VM Network. Цей сервер може бути частиною інфраструктури VirtualBox або зовнішнім відносно віртуального середовища компонентом, залежно від конфігурації мережі. DHCP-сервер не лише призначає IP-адреси, але й надає віртуальним машинам інформацію про шлюз за замовчуванням, DNS-сервери та інші мережеві параметри, необхідні для повноцінної роботи в мережі та доступу до Інтернету. Ця автоматизація суттєво спрощує налаштування мережевої взаємодії у віртуальному середовищі, особливо при роботі з великою кількістю віртуальних машин.

Підсумовуючи, представлена схема демонструє комплексну структуру віртуалізованого середовища, в якому дві віртуальні машини з Debian,

розгорнуті на платформі VirtualBox, взаємодіють між собою через віртуальну мережу, мають доступ до хост-системи через SSH та до зовнішніх мереж через підключення до Інтернету. Мережева частина забезпечує ізольовану роботу віртуальних машин з одночасним доступом до необхідних ресурсів, а наявність DHCP-сервера спрощує налаштування мережевої інфраструктури. Така архітектура надає гнучкість у використанні віртуалізованого середовища для різноманітних завдань, від навчання та тестування до розробки та розгортання серверних додатків.

2 КЛЮЧОВІ ПИТАННЯ

1. Які переваги забезпечує використання віртуалізації при вивченні методів захисту програмного забезпечення та даних у навчальному процесі?
2. Чому для створення віртуального середовища обрано саме серверний дистрибутив Debian, а не інші операційні системи?
3. У чому полягає принципова різниця між режимами мережевого адаптера NAT та «Проміжний адаптер» (Bridged Adapter) у VirtualBox, і який з них доцільно використовувати для моделювання реальних мережевих сценаріїв безпеки?
4. Які компоненти віртуалізованого середовища необхідно налаштувати для забезпечення повноцінної мережевої взаємодії між хост-системою та віртуальними машинами?
5. Яким чином здійснюється налаштування SSH-доступу до віртуальної машини, і які механізми безпеки використовує протокол SSH для захисту передачі даних?
6. Як перевірити коректність мережевої конфігурації віртуальної машини та її зв’язність із хост-системою та Інтернетом?
7. Які кроки необхідно виконати для клонування віртуальної машини в VirtualBox, і як забезпечити унікальність мережевих параметрів (IP-адреса, MAC-адреса) клонованої системи?
8. Навіщо змінювати ім’я хоста віртуальної машини, і як ця зміна впливає на ідентифікацію системи в мережі та під час SSH-підключень?
9. Яку роль відіграє DHCP-сервер у віртуалізованому середовищі, і як він сприяє автоматизації налаштування мережевих параметрів віртуальних машин?
10. Яким чином створене віртуалізоване середовище може бути використане в подальших лабораторних роботах з дисципліни «Основи захисту програмного забезпечення та даних»?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1. Встановлення програмного забезпечення для віртуалізації. Перейдіть на офіційний веб-сайт *Oracle VirtualBox* (https://www.virtualbox.org/wiki/Downloads) та завантажте інсталяційний пакет для вашої операційної системи. Встановіть основний пакет VirtualBox,

дотримуючись інструкцій інсталятора. Після завершення інсталяції основного пакету, завантажте та встановіть "*VirtualBox Extension Pack*" з того ж веб-сайту. Цей пакет розширень необхідний для підтримки USB 2.0 та 3.0 пристроїв, VirtualBox RDP, шифрування дисків та інших додаткових функцій. Для встановлення пакету розширень відкрийте VirtualBox, перейдіть в меню "Файл" → "Налаштування" → "Розширення" та додайте завантажений файл пакету розширень.

3.2. Створення віртуальної машини без жорсткого диску. Запустіть VirtualBox та натисніть кнопку "Створити". У вікні створення нової віртуальної машини введіть ім'я (наприклад, "Debian-VM1"), виберіть тип "*Linux*" та версію "*Debian (64-bit)"*. Встановіть обсяг оперативної пам'яті (рекомендовано не менше 1024 МБ) та перейдіть до налаштування жорсткого диску. Виберіть опцію "Не додавати віртуальний жорсткий диск" та підтвердіть створення віртуальної машини без диску, натиснувши "Створити". На попередження про створення віртуальної машини без жорсткого диску натисніть "Продовжити".

3.3. Додавання існуючого VDI-файлу з встановленою операційною системою. Виберіть створену віртуальну машину у списку та натисніть кнопку "Налаштування". Перейдіть у розділ "Носії інформації". У вікні, що відкрилося, виберіть контролер (IDE або SATA) та натисніть іконку "Додати дисковий привід". У діалоговому вікні виберіть опцію "Вибрати існуючий диск" та вкажіть шлях до VDI-файлу з встановленою системою Debian, який був наданий викладачем або завантажений самостійно. Підтвердіть вибір, натиснувши "Відкрити", та закрийте вікно налаштувань, натиснувши "ОК".

3.4. Налаштування мережевого адаптера. Залишаючись у вікні налаштувань віртуальної машини, перейдіть до розділу "Мережа". Для адаптера 1 змініть тип підключення з "NAT" на "Проміжний адаптер" (Bridged Adapter). У налаштуваннях проміжного адаптера виберіть фізичний мережевий інтерфейс вашої хост-системи, через який здійснюватиметься з'єднання (зазвичай це ваш основний мережевий адаптер з доступом до локальної мережі). За необхідності додайте додаткові параметри розширеного налаштування, такі як режим роботи (зазвичай має бути "Allow All"). Підтвердіть налаштування, натиснувши "ОК".

3.5. Запуск віртуальної машини та перша авторизація. Запустіть віртуальну машину, натиснувши кнопку "Запустити". Дочекайтеся завантаження операційної системи та появи запиту на введення логіна. Введіть наданий логін "*gns3*" та пароль "*gns3*" для авторизації в системі. Після успішної авторизації ви отримаєте доступ до командного рядка віртуальної машини з Debian.

3.6. Перевірка конфігурації мережного інтерфейсу. У командному рядку віртуальної машини виконайте команду ip a для відображення інформації

про мережеві інтерфейси. Знайдіть активний мережевий інтерфейс (зазвичай eth0 або enp0s3) та перевірте його IP-адресу. Переконайтеся, що IP-адреса належить до тієї ж підмережі, що і ваша хост-машина. Для порівняння можна виконати аналогічну команду на хост-машині (у Windows використовуйте *ipconfig*, у Linux/Mac - *ifconfig* або *ip a*). Якщо адреси належать до різних підмереж, перевірте налаштування мережевого адаптера у VirtualBox та наявність DHCP-сервера у вашій мережі.

3.7. Підключення до віртуальної машини через SSH. Відкрийте SSH-клієнт на вашій хост-системі. У Linux/Mac можна використовувати вбудований термінал з командою ssh, у Windows рекомендується використовувати PuTTY або інший SSH-клієнт. Введіть команду для підключення до віртуальної машини, використовуючи її IP-адресу, яку ви визначили на попередньому кроці. Наприклад: "*ssh gns3@192.168.1.X*", де X - відповідне значення з IP-адреси віртуальної машини. При першому підключенні підтвердіть автентичність хоста та введіть пароль "*gns3*" для завершення підключення.

3.8. Перевірка віддаленого доступу через SSH. Після успішного підключення через SSH перевірте можливість віддаленої роботи, виконавши кілька базових команд Linux, таких як *ls -la, pwd, whoami, hostname, uname -a.* Переконайтеся, що команди виконуються коректно та результати відображаються у вашому SSH-клієнті. Для перевірки мережевої зв'язності виконайте команду "*ping -c 4 8.8.8.8*", щоб переконатися, що віртуальна машина має доступ до Інтернету.

3.9. Зміна назви хоста віртуальної машини. Після успішного входу у систему віртуальної машини змініть назву хоста на своє прізвище або інший унікальний ідентифікатор. Для цього використовуйте команду hostnamectl з правами суперкористувача:

```
sudo hostnamectl set-hostname ivanov
```

Замініть "*ivanov*" на своє прізвище або інший унікальний ідентифікатор, який вказав викладач. Для застосування змін перезавантажте систему.

Перевірте, що зміна назви хоста відбулась успішно, виконавши команду:

```
hostname
```

Результат виконання команди повинен відображати нову назву хоста. Ця зміна допоможе ідентифікувати вашу віртуальну машину в мережі та при підключенні через SSH. Зверніть увагу, що назва хоста також буде відображатися у запрошенні командного рядка.

3.10. Коректне вимкнення віртуальної машини. Завершіть сеанс SSH, виконавши команду exit. Потім знову підключіться до віртуальної машини через SSH та виконайте команду "*sudo poweroff*" для коректного вимкнення віртуальної машини. Введіть пароль "*gns3*" для підтвердження виконання

команди з привілеями суперкористувача. Дочекайтеся повного вимкнення віртуальної машини, яке буде відображено у головному вікні VirtualBox зміною статусу машини на "Вимкнено".

3.11. Клонування віртуальної машини. У головному вікні VirtualBox переконайтеся, що ваша віртуальна машина вимкнена, виберіть її зі списку та натисніть правою кнопкою миші. У контекстному меню виберіть пункт "*Клонувати*". У вікні клонування введіть ім'я для нової віртуальної машини (наприклад, "Debian-VM2"), виберіть тип клону "Повний клон" (для створення повністю незалежної копії) та натисніть "Клонувати". Дочекайтеся завершення процесу клонування, після чого у списку віртуальних машин з'явиться новий екземпляр.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові

помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 2

Тема: Аудит безпеки Linux за допомогою Lynis

Мета роботи: Ознайомитись із принципами аудиту безпеки операційної системи Linux за допомогою інструменту Lynis, навчитися аналізувати результати аудиту та застосовувати рекомендації для підвищення рівня безпеки системи.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Концепція аудиту безпеки операційних систем

Аудит безпеки операційної системи являє собою систематичний процес оцінювання поточного стану захищеності комп'ютерної системи шляхом виявлення вразливостей, неправильних налаштувань та потенційних загроз. У сучасному світі, де кіберзагрози постійно еволюціонують, регулярне проведення аудитів безпеки стало невід'ємною складовою забезпечення інформаційної безпеки як для корпоративних інфраструктур, так і для персональних систем. Особливо важливим це є для серверних операційних систем сімейства Linux, які керують критичною інфраструктурою Інтернету, обробляють конфіденційні дані та забезпечують роботу багатьох веб-сервісів. Процес аудиту дозволяє виявити відхилення від встановлених політик безпеки, знайти застарілі компоненти програмного забезпечення, перевірити коректність налаштувань служб автентифікації та авторизації, а також оцінити загальний рівень захищеності системи.

Традиційно аудит безпеки можна поділити на кілька категорій залежно від обсягу та глибини перевірки. Базовий аудит зосереджується на перевірці критичних параметрів конфігурації, таких як налаштування файлового брандмауера, політики паролів та права доступу до системних файлів. Розширений аудит включає аналіз журналів подій, перевірку цілісності системних бінарних файлів, моніторинг мережевої активності та оцінку конфігурації всіх встановлених служб. Комплексний аудит передбачає також тестування на проникнення, аналіз вихідного коду критичних компонентів та моделювання можливих векторів атак. Для автоматизації цих процесів використовуються спеціалізовані інструменти, які здатні швидко сканувати систему, виявляти типові проблеми та надавати детальні звіти з рекомендаціями щодо усунення виявлених недоліків.

Одним із найпопулярніших інструментів для проведення аудиту безпеки в Unix-подібних системах є Lynis — відкритий сканер безпеки, розроблений компанією CISOfy. Цей інструмент відрізняється своєю універсальністю, оскільки підтримує широкий спектр дистрибутивів Linux, включаючи Debian, Ubuntu, Red Hat, CentOS, а також операційні системи BSD та macOS. Lynis працює за принципом не інвазивного сканування, тобто не вносить жодних змін до системи під час аудиту, а лише зчитує конфігураційні файли, перевіряє стан

служб та аналізує системні параметри. Інструмент виконує понад триста різноманітних тестів, кожен з яких перевіряє конкретний аспект безпеки, і класифікує результати за рівнем критичності. Завдяки модульній архітектурі Lynis може бути розширений додатковими плагінами для перевірки специфічних конфігурацій або галузевих стандартів безпеки.

1.2 Архітектура та принципи роботи Lynis

Lynis побудований як скриптова система на мові Bash, що робить його легким у розгортанні та не потребує складної інсталяції залежностей. Під час запуску аудиту інструмент послідовно виконує набір тестів, організованих у логічні групи відповідно до компонентів операційної системи. Спочатку перевіряється завантажувач системи, зокрема конфігурація GRUB, наявність захисту паролем для режиму відновлення та коректність параметрів ядра. Далі аналізується конфігурація самого ядра Linux, включаючи активовані модулі безпеки, параметри мережевого стеку, налаштування захисту від переповнення буфера та інші критичні системні налаштування. Наступним етапом є перевірка служб автентифікації, де Lynis аналізує файли */etc/passwd*, */etc/shadow*, налаштування PAM (Pluggable Authentication Modules) та політики паролів, визначені в */etc/login.defs* та */etc/security/pwquality.conf*.

Особливу увагу інструмент приділяє мережевим службам, оскільки саме вони найчастіше стають точками входу для зловмисників. Lynis сканує активні порти, перевіряє конфігурацію SSH-сервера у файлі */etc/ssh/sshd_config*, аналізує налаштування веб-серверів Apache або Nginx, баз даних MySQL або PostgreSQL, поштових серверів та інших мережевих демонів. Для кожної служби перевіряється використання шифрування, наявність обмежень доступу, правильність налаштувань логування та відповідність рекомендованим практикам безпеки. Після аналізу мережевих служб Lynis переходить до перевірки файлової системи, де досліджує права доступу до критичних каталогів, наявність файлів із небезпечними дозволами, коректність налаштувань монтування файлових систем та використання механізмів шифрування дисків.

Результати кожного тесту класифікуються за допомогою кольорового кодування та текстових міток. Успішно пройдені перевірки позначаються зеленим кольором та міткою [OK], що свідчить про відповідність даного параметра рекомендованим налаштуванням безпеки. Попередження, позначені жовтим кольором та міткою [WARNING], вказують на потенційні проблеми, які не є критичними, але можуть знизити загальний рівень безпеки системи. Серйозні проблеми позначаються червоним кольором, хоча у стандартному режимі роботи Lynis рідко використовує цей рівень критичності для збереження конструктивності звіту. Окрім безпосередніх результатів тестів, інструмент генерує секцію Suggestions, де містяться конкретні рекомендації щодо покращення конфігурації, та секцію Warnings, де зібрані найбільш важливі виявлені проблеми, що потребують негайного вирішення.

1.3 Метрики оцінювання та інтерпретація результатів аудиту

Після завершення всіх тестів Lynis обчислює загальний показник захищеності системи, який називається Hardening Index. Цей індекс виражається у відсотках і представляє собою співвідношення кількості успішно пройдених тестів до загальної кількості виконаних перевірок. Значення індексу від нуля до п’ятдесяти відсотків вказує на критично низький рівень захищеності, коли система має численні серйозні вразливості та неправильні налаштування, що робить її легкою ціллю для атак. Індекс у діапазоні від п’ятдесяти до семидесяти відсотків свідчить про базовий рівень безпеки, достатній для навчальних або тестових середовищ, але неприйнятний для продакшн-систем, які обробляють реальні дані. Значення від семидесяти до дев’яноста відсотків вважається хорошим рівнем для більшості комерційних застосувань, тоді як індекс понад дев’яносто відсотків досягається лише за умови комплексного підходу до безпеки та ретельного налаштування всіх компонентів системи.

Важливо розуміти, що Hardening Index є відносною метрикою і не може розглядатися як абсолютна гарантія безпеки системи. Високий індекс свідчить про те, що система відповідає багатьом загальноприйнятим практикам безпеки, але не враховує специфічних загроз, характерних для конкретного сценарію використання. Наприклад, система з індексом дев’яносто п’ять відсотків все ще може бути вразливою до нульових днів у встановленому програмному забезпеченні або до атак соціальної інженерії на користувачів. Тому аудит безпеки за допомогою Lynis слід розглядати як один із елементів комплексної стратегії захисту, яка також включає регулярне оновлення програмного забезпечення, моніторинг безпеки в режимі реального часу, навчання персоналу та розробку процедур реагування на інциденти.

Аналіз результатів аудиту вимагає розуміння контексту роботи системи та пріоритизації виявлених проблем. Не всі рекомендації Lynis однаково важливі для кожної конфігурації, і деякі з них можуть навіть суперечити функціональним вимогам до системи. Наприклад, рекомендація щодо відключення SSH-доступу з використанням пароля може бути критично важливою для серверів, доступних з Інтернету, але неприйнятною для локальних систем розробки, де зручність доступу має пріоритет над максимальною безпекою. Подібним чином, вимога щодо шифрування всіх розділів диска може бути обов’язковою для ноутбуків, які можуть бути втрачені або викрадені, але необов’язковою для стаціонарних серверів у фізично захищених дата-центрах. Тому кожну рекомендацію необхідно оцінювати з урахуванням моделі загроз, специфічної для конкретного середовища, та балансу між безпекою, продуктивністю та зручністю експлуатації.

1.4 Основні команди та їх застосування в процесі аудиту

Робота з інструментом Lynis та аналіз його результатів передбачає використання базових команд операційної системи Linux для встановлення програмного забезпечення, виконання аудиту та обробки згенерованих звітів. Нижче наведено таблицю з основними командами, які застосовуються під час виконання лабораторної роботи, разом із детальними поясненнями їх призначення та параметрів.

Таблиця 2.1 – Команди для роботи з Lynis та аналізу результатів аудиту

Команда	Опис та призначення
sudo apt update	Оновлення локальних списків доступних пакетів з репозиторіїв Debian. Команда завантажує метадані про найновіші версії пакетів, але не встановлює оновлення.
sudo apt install lynis	Встановлення пакету Lynis з офіційного репозиторію Debian. Менеджер пакетів автоматично вирішує залежності та завантажує необхідні файли.
sudo lynis --version	Перевірка встановленої версії Lynis. Команда виводить номер версії інструменту, що дозволяє переконатися у коректності інсталяції.
sudo lynis audit system	Запуск повного аудиту безпеки системи. Команда виконує всі доступні тести та виводить результати у термінал з кольоровим кодуванням.
sudo lynis audit system --report-file <шлях>	Виконання аудиту з збереженням структурованого звіту у бінарному форматі Lynis. Файл містить детальну інформацію про кожен тест у машиночитному форматі.
sudo lynis audit system | tee <файл>	Запуск аудиту з одночасним збереженням виводу в текстовий файл. Команда tee дублює вивід у термінал та файл, зберігаючи повний лог аудиту.
nano <файл>	Відкриття текстового файлу в редакторі nano для перегляду та аналізу. Зручний для читання довгих звітів завдяки можливості прокручування та пошуку.
grep "WARNING" <файл>	Пошук рядків, що містять слово “WARNING”, у файлі звіту. Дозволяє швидко виявити всі попередження, згенеровані під час аудиту.
grep "Suggestion" <файл>	Фільтрування рекомендацій зі звіту. Команда виводить лише ті рядки, які містять конкретні пропозиції щодо покращення безпеки.
cat <файл> | less	Постраничний перегляд вмісту файлу. Команда less дозволяє зручно гортати великі текстові файли, використовуючи клавіші стрілок та функції пошуку.

Використання цих команд у правильній послідовності забезпечує ефективне проведення аудиту безпеки та якісний аналіз отриманих результатів. Спочатку необхідно оновити списки пакетів командою sudo apt update, що гарантує встановлення актуальної версії Lynis без застарілих залежностей. Після встановлення інструменту за допомогою *sudo apt install lynis* слід перевірити версію командою *sudo lynis --version*, щоб переконатися у коректності інсталяції та відповідності очікуваній версії програми. Запуск базового аудиту виконується командою sudo lynis audit system, яка ініціює послідовне тестування всіх компонентів системи та виводить результати

безпосередньо у термінал з використанням кольорового кодування для візуального виділення проблем.

Для збереження результатів аудиту використовуються дві паралельні стратегії, кожна з яких має свої переваги. Команда з параметром --report-file генерує структурований звіт у власному форматі Lynis, який зручний для автоматичної обробки скриптами або інтеграції з системами моніторингу безпеки. Водночас команда з використанням tee створює повний текстовий лог усього виводу терміналу, включаючи всі діагностичні повідомлення, прогрес-бари та додаткову інформацію, яка може бути корисною при глибокому аналізі або відтворенні проблем. Обидва файли доповнюють один одного і разом забезпечують комплексну документацію процесу аудиту, необхідну як для поточного аналізу, так і для майбутніх порівняльних досліджень ефективності впроваджених заходів безпеки.

Після збереження звітів ключовим етапом стає їх детальний аналіз за допомогою команд фільтрування та пошуку. Команда grep дозволяє швидко ізолювати найбільш важливі частини звіту, такі як попередження та рекомендації, без необхідності ручного прокручування тисяч рядків тексту. Використання nano або less забезпечує зручний інтерфейс для навігації великими файлами звітів, де можна використовувати функції пошуку для швидкого переходу до конкретних секцій або кодів рекомендацій. Комбінування цих інструментів із знанням структури звітів Lynis формує ефективний робочий процес, який дозволяє системним адміністраторам швидко ідентифікувати критичні проблеми безпеки та розробляти плани їх усунення на основі пріоритетів і доступних ресурсів.

Важливо розуміти, що команди, наведені в таблиці, представляють лише базовий набір операцій, необхідних для виконання стандартного аудиту безпеки. Lynis підтримує значно ширший спектр параметрів і режимів роботи, включаючи вибіркове виконання окремих груп тестів, інтеграцію з зовнішніми системами збору логів, генерацію звітів у форматі JSON для автоматичного парсингу та багато інших можливостей. Глибоке вивчення документації інструменту та експерименти з різними режимами його роботи дозволяють адаптувати процес аудиту до специфічних вимог конкретного середовища та отримати максимальну користь від цього потужного інструменту безпеки. Регулярне проведення аудитів з використанням Lynis та послідовне впровадження його рекомендацій формує культуру безпеки в організації та значно знижує ризики успішних кібератак на інфраструктуру.

2 КЛЮЧОВІ ПИТАННЯ

1. У чому полягає мета та основні етапи аудиту безпеки операційної системи, і як він допомагає виявити вразливості та неправильні налаштування?
2. Які переваги надає використання автоматизованого інструменту, такого як Lynis, порівняно з ручним аналізом конфігурації системи?

3. Які основні категорії безпеки (наприклад, автентифікація, мережева безпека, файлова система) перевіряє Lynis, і чому кожна з них є критичною для загального рівня захисту?

4. Як інтерпретувати результати аудиту Lynis: у чому різниця між «Warnings» (попередженнями) та «Suggestions» (рекомендаціями), і які з них мають вищий пріоритет при усуненні?

5. Що таке Hardening Index, як він розраховується, і які значення цього показника вважаються задовільними для навчального, тестового та продакшн-середовища?

6. Чому важливо зберігати звіт аудиту у двох форматах (текстовий лог та структурований .dat-файл), і як кожен із цих форматів може бути використаний на практиці?

7. Які типові проблеми безпеки може виявити Lynis у стандартній інсталяції Debian (наприклад, слабкі налаштування SSH, відсутність брандмауера, неправильні права доступу), і як їх усунути?

8. Як правильно дослідити конкретну рекомендацію Lynis (наприклад, AUTH-9328): де знайти додаткову інформацію, як зрозуміти суть проблеми та які кроки необхідно виконати для її вирішення?

9. Чи гарантує високий Hardening Index абсолютну безпеку системи? Поясніть, чому аудит за допомогою Lynis є лише одним із компонентів комплексної стратегії кібербезпеки.

10. Як можна інтегрувати Lynis у процес постійного моніторингу безпеки, наприклад, за допомогою планувальника cron, і як це допомагає підтримувати відповідність стандартам (наприклад, CIS Benchmarks)?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1. Запуск віртуальної машини. Перед початком аудиту безпеки необхідно переконатися, що робоче середовище готове до виконання завдання. У цьому випадку таким середовищем є віртуальна машина з операційною системою Debian Linux 12, створена та налаштована в рамках попередньої лабораторної роботи. Запуск віртуальної машини через гіпервізор VirtualBox забезпечує ізольоване та контролюване середовище, у якому можна безпечно виконувати аналіз безпеки, не створюючи ризиків для основної системи. Після завантаження ОС студент повинен увійти до системи, використовуючи облікові дані, надані викладачем, щоб отримати доступ до терміналу та необхідних привілеїв адміністратора.

Запустіть програму Oracle VirtualBox. У головному вікні знайдіть віртуальну машину з Debian Linux 12. Виділіть її та натисніть кнопку «Запустити». Дочекайтеся повного завантаження операційної системи. Увійдіть до системи, використовуючи логін *gns3* та пароль *gns3*.

Після успішного входу ви повинні побачити запрошення командного рядка, що свідчить про готовність системи до подальшої роботи. Переконайтеся, що у вас є доступ до мережі (наприклад, за допомогою команди

ping -c 3 8.8.8.8), оскільки встановлення програмного забезпечення потребує підключення до Інтернету.

3.2. Встановлення інструменту Lynis. Для проведення аудиту безпеки використовується спеціалізований інструмент Lynis — відкритий сканер безпеки, призначений для аналізу конфігурації систем Unix/Linux. Він перевіряє сотні параметрів, включаючи налаштування автентифікації, мережевих служб, файлових систем, журналів подій тощо. Оскільки Lynis не входить до базового набору пакетів Debian, його необхідно встановити з офіційного репозиторію. Це робиться за допомогою менеджера пакетів apt, який забезпечує автоматичне вирішення залежностей та перевірку цілісності пакетів. Перед встановленням будь-якого програмного забезпечення рекомендується оновити локальні списки пакетів, щоб гарантувати встановлення найновішої версії.

```
sudo apt update
sudo apt install lynis
```

Під час встановлення система може запитати підтвердження — введіть Y та натисніть Enter. Після завершення встановлення перевірте версію інструменту, щоб переконатися у його коректній інсталяції:

```
sudo lynis --version
```

У виводі має відображатися номер версії (наприклад, 3.0.9). Якщо команда повертає помилку або не знаходить програму, повторіть встановлення, переконавшись у наявності мережевого з’єднання та правильності джерел пакетів у */etc/apt/sources.list.*

3.3. Запуск аудиту безпеки. Основна функція Lynis — це виконання комплексного аудиту безпеки системи. Команда lynis audit system ініціює послідовний аналіз усіх ключових компонентів ОС: від завантажувача GRUB до конфігурації мережевих служб. Протягом аудиту інструмент виконує сотні тестів, класифікуючи результати за рівнем важливості: зелені рядки — успішні перевірки, жовті — рекомендації (Suggestions), червоні — попередження (Warnings). Цей процес може тривати від кількох хвилин до півгодини, залежно від продуктивності системи та кількості встановлених служб. Аудит проводиться з правами суперкористувача, оскільки багато конфігураційних файлів та системних параметрів доступні лише root.

```
sudo lynis audit system
```

Під час виконання уважно спостерігайте за виводом у терміналі. Зверніть увагу на червоні та жовті повідомлення — саме вони містять інформацію про потенційні проблеми безпеки. Після завершення аудиту Lynis виведе загальний

показник Hardening index — це узагальнена оцінка рівня захищеності системи у відсотках. Значення понад 70% вважається задовільним для навчального середовища, але для продакшн-систем рекомендується прагнути до 90% і вище.

3.4. Збереження звіту аудиту. Для подальшого аналізу та включення до звіту необхідно зберегти результати аудиту у файл. Lynis підтримує кілька форматів звітів, але для навчальних цілей достатньо текстового файлу, який легко читати та аналізувати. Спочатку створюється окрема директорія ~/lynis-reports для зберігання всіх матеріалів, пов'язаних із аудитом. Потім аудит запускається повторно, але з додатковим параметром --report-file, який вказує шлях до бінарного файлу звіту. Одночасно використовується команда tee, щоб зберегти читабельний текстовий варіант звіту, який містить увесь вивід терміналу, включно з кольоровим кодуванням (у вигляді ANSI-кодів).

```
mkdir ~/lynis-reports
sudo lynis audit system --report-file ~/lynis-reports/audit-
report.dat
sudo lynis audit system | tee ~/lynis-reports/audit-report.txt
```

Після виконання цих команд у каталозі ~/lynis-reports/ з'являться два файли: audit-report.dat (структурований звіт у внутрішньому форматі Lynis) та audit-report.txt (повний текстовий лог аудиту). Останній файл слід використовувати для аналізу, оскільки він містить усі повідомлення, включаючи Suggestions та Warnings, у зручному для читання вигляді.

3.5. Аналіз результатів аудиту. Аналіз звіту є ключовим етапом лабораторної роботи, оскільки саме на цьому етапі формується розуміння стану безпеки системи. Особливу увагу слід приділити двом секціям: «Warnings» — тут наведено критичні або серйозні проблеми, які можуть призвести до компрометації системи, і «Suggestions» — рекомендації щодо покращення конфігурації, які не є критичними, але підвищують загальний рівень захисту. Кожна рекомендація має унікальний ідентифікатор (наприклад, AUTH-9328), що полегшує пошук додаткової інформації. Також у кінці звіту вказується Hardening index — числовий показник, який дозволяє оцінити ефективність заходів безпеки.

```
nano ~/lynis-reports/audit-report.txt
```

Прокрутіть файл до кінця, знайдіть секції Warnings та Suggestions. Зафіксуйте загальну кількість рекомендацій та попереджень, а також значення Hardening index. Ці дані будуть використані в звіті. Зверніть увагу, що деякі рекомендації можуть стосуватися відсутності брандмауера, слабких налаштувань SSH, відсутності оновлень безпеки або неправильних прав доступу до системних файлів.

3.6. Дослідження конкретної рекомендації. Кожен студент аналізує одну рекомендацію зі списку Suggestions, номер якої відповідає його порядковому номеру в журналі групи. Це дозволяє розподілити навантаження та забезпечити глибоке розуміння різних аспектів безпеки. Для виконання цього кроку необхідно знайти відповідну рекомендацію у звіті, записати її код та повний опис, а потім дослідити її суть за допомогою офіційної документації Lynis, довідників Debian або авторитетних джерел у сфері кібербезпеки.

У файлі *audit-report.txt* знайдіть секцію Suggestions. Порахуйте рекомендації зверху вниз і виберіть ту, що відповідає вашому номеру в журналі. Запишіть її унікальний код (наприклад, SSH-7408) та повний текст опису.

Цей крок формує основу для подальшого практичного завдання — застосування рекомендації. Тому важливо не лише скопіювати текст, але й зрозуміти, яку саме проблему виявив Lynis і чому вона є значущою з точки зору безпеки.

3.7. Застосування рекомендації. Останній крок передбачає не просто теоретичне розуміння, а й практичне застосування знань. На основі дослідженої рекомендації студент повинен знайти або розробити конкретні кроки для усунення виявленої проблеми або покращення конфігурації. Наприклад, якщо рекомендація стосується відключення небезпечного алгоритму шифрування в SSH, слід вказати, який параметр змінити у файлі */etc/ssh/sshd_config*, як перезапустити службу та як перевірити результат. Це розвиває навички самостійного пошуку рішень, роботи з документацією та впровадження заходів безпеки.

Використовуйте пошукові системи, офіційну документацію Lynis (*https://cisofy.com/documentation/lynis/)* та довідники Debian. Знайдіть конкретні команди, зміни конфігураційних файлів або інші дії, необхідні для виконання рекомендації. Запишіть ці кроки у звіт, вказавши суть проблеми, можливі наслідки її ігнорування та детальний план усунення.

Навіть якщо ви не виконуєте зміни реально (через обмеження навчального середовища), опис має бути технічно точним і відтворюваним. Це демонструє професійне розуміння механізмів захисту в Linux.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних

цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 3

Тема: Захист від атак грубої сили за допомогою Fail2Ban

Мета роботи: набуття студентами практичних навичок з налаштування системи захисту серверів від атак методом грубої сили шляхом встановлення, конфігурування та тестування Fail2Ban у середовищі віртуальної машини.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Теоретичні основи захисту від атак методом грубої сили та архітектура системи Fail2Ban

У сучасному світі інформаційних технологій безпека комп'ютерних систем та серверів становить одну з найважливіших проблем, з якою стикаються як великі підприємства, так і маленькі організації. Постійне зростання кількості та складності кібератак вимагає впровадження ефективних систем захисту інформації та своєчасного реагування на потенційні загрози. Особливо вразливими є сервери з відкритими портами для надання мережевих послуг, таких як SSH (Secure Shell), FTP (File Transfer Protocol), HTTP (HyperText Transfer Protocol) та інші, які можуть зазнавати атак методом грубої сили (brute force), підбору паролів за словником (dictionary attack) та інших видів несанкціонованого доступу. Незважаючи на використання стійких криптографічних алгоритмів та протоколів у таких сервісах, як SSH, який шифрує весь трафік між клієнтом і сервером, захист системи аутентифікації залишається критичним питанням, оскільки слабкі або скомпрометовані паролі можуть призвести до несанкціонованого доступу до системи. Сучасні методи захисту включають не лише використання сильних паролів та ключів шифрування, але й механізми виявлення підозрілої активності, блокування підозрілих IP-адрес та сповіщення адміністраторів про потенційні загрози безпеці.

Атаки методом грубої сили є одним із найпоширеніших векторів атак на серверні системи, особливо на сервіси віддаленого доступу, такі як SSH, FTP та веб-додатки. Сутність таких атак полягає у систематичному переборі всіх можливих комбінацій символів для підбору пароля або ключа шифрування. Зловмисники використовують спеціалізоване програмне забезпечення, яке автоматично генерує та випробовує різні комбінації, намагаючись отримати несанкціонований доступ до системи. Успішність атаки методом грубої сили зазвичай залежить від складності паролів, часу, доступного зловмисникам, та обчислювальних ресурсів, які вони можуть використати. В сучасних умовах, коли обчислювальна потужність комп'ютерів постійно зростає, а доступ до розподілених обчислювальних ресурсів стає дешевшим, ризик успішної атаки методом грубої сили значно збільшується. Крім того, атаки можуть бути модифіковані для включення елементів соціальної інженерії, використання словників паролів, та інших методів, що підвищують їхню ефективність

порівняно з простим перебором. Для захисту від таких атак організації впроваджують політики безпеки, які вимагають використання складних паролів, двофакторної аутентифікації, та обмеження кількості спроб входу в систему.

Системи виявлення та запобігання вторгнень (Intrusion Detection and Prevention Systems, IDPS) є критичними компонентами сучасної інфраструктури інформаційної безпеки, які дозволяють ідентифікувати потенційні атаки та запобігати їм в режимі реального часу. На відміну від традиційних методів захисту, таких як паролі та шифрування, IDPS аналізують мережевий трафік та системні події, шукаючи підозрілі шаблони активності, що можуть вказувати на спроби несанкціонованого доступу або інші види кібератак. Системи виявлення вторгнень (Intrusion Detection Systems, IDS) фокусуються на моніторингу та сповіщенні про підозрілу активність, тоді як системи запобігання вторгнень (Intrusion Prevention Systems, IPS) додатково здатні автоматично вживати захисних заходів для блокування атак. Ці системи можуть базуватися на різних методологіях, включаючи аналіз на основі сигнатур, який порівнює активність з відомими шаблонами атак, та аналіз аномалій, який виявляє відхилення від нормальної поведінки системи. Ефективне використання IDPS вимагає регулярного оновлення бази сигнатур атак, настройки параметрів виявлення для мінімізації помилкових спрацьовувань та інтеграції з іншими компонентами інфраструктури безпеки для забезпечення комплексного захисту.

Fail2Ban є потужною системою запобігання вторгнень з відкритим кодом, розробленою спеціально для захисту Linux-серверів від атак методом грубої сили та інших типів автоматизованих кібератак. Основний принцип роботи Fail2Ban базується на моніторингу файлів журналів (логів) системних сервісів, таких як SSH, FTP, HTTP та інші, для виявлення підозрілих шаблонів активності, таких як багаторазові невдалі спроби аутентифікації. При виявленні такої активності Fail2Ban автоматично модифікує правила фаєрволу (найчастіше використовується iptables в Linux системах) для блокування IP-адреси, з якої надходять підозрілі запити, на певний період часу, що значно ускладнює проведення атак методом грубої сили. Архітектура Fail2Ban включає гнучку систему фільтрів, що використовують регулярні вирази для аналізу записів у файлах журналів, та набір дій, які можуть бути виконані при виявленні підозрілої активності. Крім блокування IP-адрес, Fail2Ban може відправляти повідомлення адміністраторам, виконувати користувацькі скрипти та здійснювати інші дії, що робить його потужним та адаптивним інструментом для захисту серверів. Fail2Ban підтримує широкий спектр сервісів та може бути легко налаштований для роботи з нестандартними конфігураціями через систему "в'язниць" (jails) - окремих конфігураційних блоків для кожного сервісу, що підлягає захисту.

Fail2Ban має модульну архітектуру, яка складається з кількох ключових компонентів, що взаємодіють між собою для забезпечення ефективного захисту серверів від атак методом грубої сили, як показано на рисунку архітектури системи. Центральним компонентом є демон fail2ban-server, який працює в

фоновому режимі та координує роботу всіх інших компонентів системи. Цей демон відповідає за моніторинг файлів журналів, застосування фільтрів для виявлення підозрілої активності та виконання відповідних дій відповідно до налаштованих правил. Іншим важливим компонентом є утиліта fail2ban-client, яка надає інтерфейс командного рядка для управління демоном fail2ban-server, дозволяючи адміністраторам переглядати статус системи, змінювати конфігурацію "в'язниць" та виконувати інші адміністративні завдання. Конфігурація Fail2Ban зберігається в наборі файлів в директорії /etc/fail2ban/, де файл jail.conf (або jail.local) містить загальні налаштування системи та конфігурації окремих "в'язниць", а директорія filter.d містить фільтри, що визначають, які шаблони в файлах журналів будуть розглядатися як підозрілі. Директорія action.d містить шаблони дій, які можуть бути виконані при виявленні підозрілої активності, таких як модифікація правил фаєрволу або відправка сповіщень. Взаємодія між цими компонентами дозволяє Fail2Ban ефективно виявляти та блокувати потенційні атаки, адаптуватися до різних сценаріїв використання та інтегруватися з іншими компонентами інфраструктури безпеки.

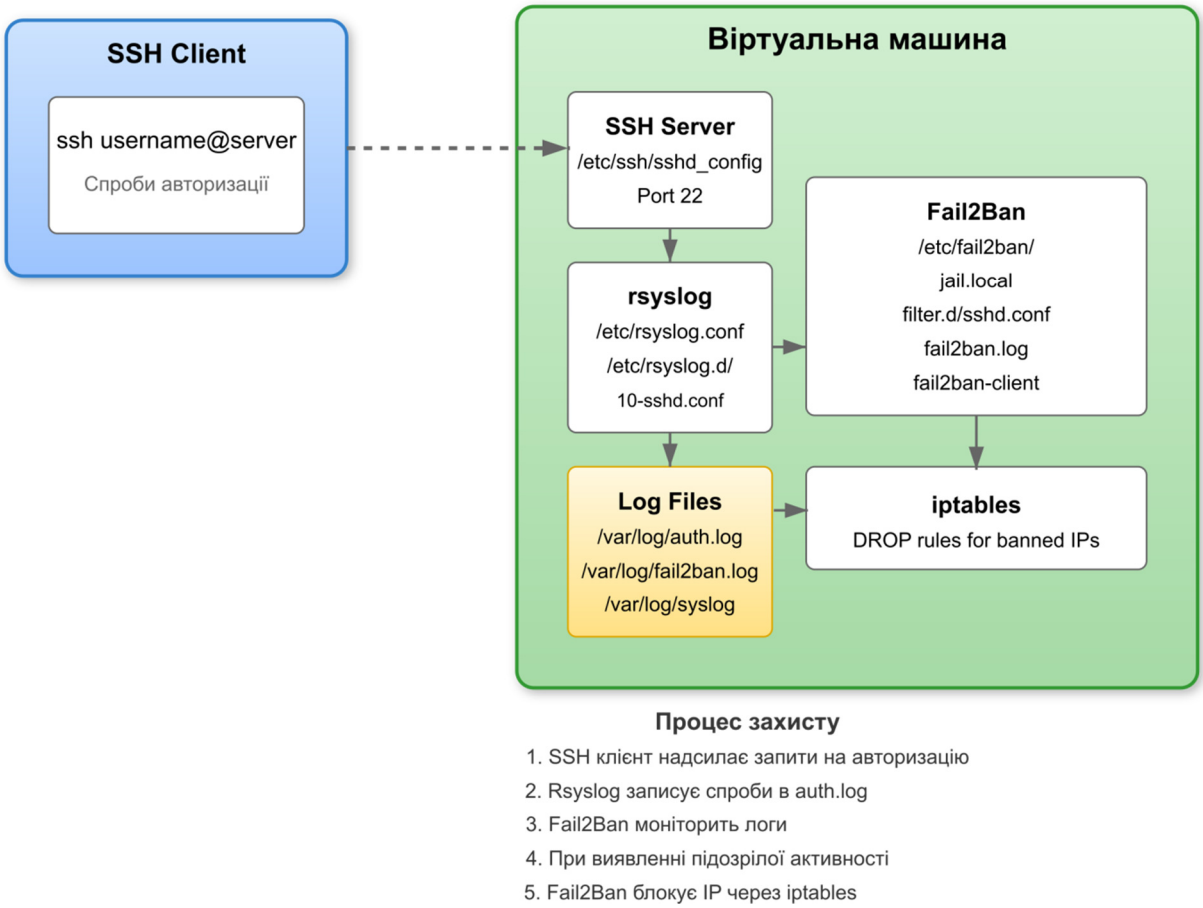

Рисунок 3.1 – Модульна архітектура Fail2Ban

Механізм роботи Fail2Ban в контексті захисту SSH-сервера від атак методом грубої сили демонструє ефективність цієї системи для забезпечення безпеки серверів. Як показано на діаграмі взаємодії компонентів системи,

процес починається з того, що SSH-клієнт з віддаленої машини надсилає запит на автентифікацію до SSH-сервера на цільовій машині. SSH-сервер обробляє цей запит, перевіряючи надані облікові дані (логін та пароль або ключ) та, залежно від результату перевірки, надає доступ або відмовляє в ньому. Важливо, що всі спроби автентифікації, успішні та невдалі, реєструються в системних журналах, зокрема в файлі /var/log/auth.log. Сервіс rsyslog, який є стандартною системою логування в більшості дистрибутивів Linux, відповідає за запис цих подій у відповідні файли журналів. Саме ці записи стають об'єктом моніторингу для Fail2Ban. Fail2Ban використовує спеціально налаштовані фільтри, які містять регулярні вирази для ідентифікації записів про невдалі спроби автентифікації в файлах журналів. Коли кількість невдалих спроб з однієї IP-адреси перевищує встановлений поріг (параметр maxretry у конфігурації Fail2Ban), система ініціює блокування цієї IP-адреси. Блокування здійснюється шляхом додавання правила DROP для цієї IP-адреси до ланцюжка правил фаєрволу iptables, що ефективно блокує всі подальші з'єднання з цієї адреси протягом визначеного періоду часу (параметр bantime). По закінченні цього періоду правило автоматично видаляється, дозволяючи повторні спроби з'єднання.

1.2 Налаштування та конфігурація Fail2Ban

Налаштування та конфігурація Fail2Ban є важливими аспектами для забезпечення ефективного захисту серверів від атак методом грубої сили, оскільки оптимальні параметри можуть значно підвищити рівень безпеки системи. Базова конфігурація Fail2Ban зберігається у файлі /etc/fail2ban/jail.conf, але для забезпечення збереження налаштувань при оновленні пакету рекомендується створювати окремий файл jail.local, який має пріоритет над стандартними налаштуваннями. В конфігураційному файлі jail.local адміністратор може визначити загальні параметри, такі як ignoreip (список IP-адрес, які ніколи не будуть заблоковані), findtime (період часу, протягом якого підраховуються невдалі спроби автентифікації), maxretry (максимальна кількість невдалих спроб перед блокуванням), та bantime (тривалість блокування IP-адреси). Крім загальних параметрів, jail.local містить конфігурації окремих "в'язниць" для різних сервісів, таких як [sshd] для SSH-сервера, [apache] для веб-сервера Apache та інші, кожна з яких може мати власні специфічні налаштування. Для кожної "в'язниці" можна вказати, які фільтри використовувати (параметр filter), які файли журналів моніторити (logpath), які порти захищати (port), та які дії виконувати при виявленні підозрілої активності (action). Фільтри, які використовують регулярні вирази для ідентифікації підозрілих записів у файлах журналів, зберігаються в директорії /etc/fail2ban/filter.d, а шаблони дій - в директорії /etc/fail2ban/action.d. Налаштування цих компонентів дозволяє адаптувати Fail2Ban до специфічних потреб кожного сервера та забезпечити оптимальний баланс між безпекою та доступністю сервісів.

Хоча Fail2Ban часто асоціюється із захистом SSH-серверів, його можливості значно ширші, і він може бути ефективно використаний для захисту різноманітних мережевих сервісів від атак методом грубої сили та інших типів автоматизованих атак. Наприклад, для веб-серверів, таких як Apache або Nginx, Fail2Ban може моніторити файли журналів для виявлення спроб підбору паролів до адміністративних інтерфейсів, сканування вразливостей, або спроб використання відомих експлойтів. Для поштових серверів, таких як Postfix або Dovecot, Fail2Ban може блокувати IP-адреси, з яких надходять численні невдалі спроби авторизації в SMTP або IMAP/POP3. Крім стандартних сервісів, Fail2Ban може бути налаштований для захисту практично будь-якого сервісу, який генерує логи з інформацією про спроби автентифікації або інші підозрілі активності, через створення відповідних фільтрів та конфігурацій "в'язниць". Розширені можливості Fail2Ban включають інтеграцію з іншими системами безпеки, такими як систем виявлення вторгнень Snort або Suricata, для створення багаторівневого захисту. Також Fail2Ban може бути налаштований для відправки сповіщень адміністраторам через електронну пошту, месенджери або інші канали комунікації, дозволяючи оперативно реагувати на потенційні загрози. Додатково, Fail2Ban підтримує використання RDP (Remote Desktop Protocol) для блокування атак на Windows-сервери, що робить його універсальним інструментом для захисту гетерогенних середовищ. Інтеграція Fail2Ban з системами моніторингу, такими як Nagios або Zabbix, дозволяє включити інформацію про спроби атак та блокування в загальну картину стану інфраструктури.

Для забезпечення максимальної ефективності захисту серверів за допомогою Fail2Ban необхідно регулярно аналізувати його роботу та оптимізувати налаштування відповідно до специфіки конкретної інфраструктури та поточних векторів атак. Аналіз ефективності Fail2Ban включає моніторинг файлів журналів самого Fail2Ban (/var/log/fail2ban.log), які містять інформацію про виявлені підозрілі активності, заблоковані IP-адреси та інші системні події. Крім того, важливо аналізувати журнали захищених сервісів для виявлення потенційних пропущених атак або помилкових спрацьовувань, які можуть вказувати на необхідність коригування фільтрів або інших параметрів. Оптимізація роботи Fail2Ban може включати коригування параметрів maxretry та findtime для балансування між безпекою та зручністю користувачів, оскільки надто жорсткі налаштування можуть призвести до блокування легітимних користувачів, а надто м'які - до пропуску реальних атак. Також важливо оптимізувати параметр bantime, встановлюючи його достатньо довгим для ефективного захисту від атак методом грубої сили, але не настільки довгим, щоб створювати надмірні незручності для легітимних користувачів у випадку помилкових спрацьовувань. Для випадків, коли стандартні фільтри не ефективно виявляють всі спроби атак, може бути необхідним створення або модифікація фільтрів з використанням регулярних виразів, що більш точно відповідають специфічним шаблонам атак, які спостерігаються в конкретній

інфраструктурі. Також рекомендується регулярно оновлювати Fail2Ban до останньої версії для отримання нових функцій та виправлень безпеки.

Хоча Fail2Ban є ефективним інструментом для захисту від атак методом грубої сили, комплексна безпека серверів вимагає застосування додаткових заходів захисту, які доповнюють його функціональність та компенсують потенційні обмеження. Одним з ключових додаткових заходів є впровадження сильних політик паролів, які вимагають використання складних паролів, регулярної їх зміни та унеможливлюють повторне використання старих паролів. Ще більш ефективним є перехід від парольної аутентифікації до аутентифікації на основі ключів для сервісів, таких як SSH, що значно підвищує рівень безпеки, оскільки ключі шифрування є значно стійкішими до підбору, ніж паролі. Впровадження двофакторної аутентифікації (2FA) для критичних сервісів та систем додає додатковий рівень захисту, вимагаючи від користувачів не лише знання пароля, але й володіння певним фізичним пристроєм або доступу до певного сервісу для підтвердження особистості. Регулярний аудит безпеки, включаючи сканування вразливостей та тестування на проникнення, дозволяє виявляти та усувати потенційні вразливості до того, як вони можуть бути використані зловмисниками. Впровадження систем виявлення та запобігання вторгнень на рівні мережі (NIDS/NIPS) та на рівні хоста (HIDS/HIPS) забезпечує багаторівневий захист та дозволяє виявляти більш широкий спектр потенційних атак, ніж лише атаки методом грубої сили. Регулярне оновлення програмного забезпечення та операційних систем для усунення відомих вразливостей та обмеження мережевого доступу до критичних сервісів через фаєрволи та списки контролю доступу (ACL) є також важливими компонентами комплексної стратегії безпеки.

У сучасних умовах постійного зростання кількості та складності кібератак, системи захисту від атак методом грубої сили, такі як Fail2Ban, стають необхідними компонентами інфраструктури безпеки для будь-якої організації, яка надає мережеві сервіси. Ефективність Fail2Ban базується на його здатності автоматично виявляти підозрілу активність та блокувати потенційно шкідливі IP-адреси без необхідності втручання адміністраторів, що дозволяє оперативно реагувати на атаки в режимі реального часу. Однак важливо розуміти, що Fail2Ban не є панацеєю від усіх типів кібератак та має бути лише одним з компонентів комплексної стратегії безпеки, яка включає сильні політики паролів, аутентифікацію на основі ключів, двофакторну аутентифікацію, регулярний аудит безпеки та інші заходи захисту. Перспективи розвитку систем захисту від атак методом грубої сили включають інтеграцію з системами штучного інтелекту та машинного навчання для покращення точності виявлення потенційних атак та зменшення кількості помилкових спрацьовувань. Також очікується більш тісна інтеграція з хмарними сервісами та контейнерними платформами, такими як Docker та Kubernetes, для забезпечення ефективного захисту в сучасних динамічних інфраструктурах. Розвиток стандартів та протоколів обміну інформацією про загрози між різними системами безпеки дозволить створювати більш ефективні рішення для виявлення та запобігання скоординованим атакам, які можуть використовувати

multiple IP-адреси для обходу традиційних методів захисту. В цілому, незважаючи на постійну еволюцію методів атак, Fail2Ban та подібні системи залишаються критично важливими інструментами для захисту від одного з найпоширеніших видів кібератак - атак методом грубої сили.

Таблиця 3.1 містить 30 варіантів налаштувань параметрів Fail2Ban для виконання лабораторної роботи. Кожен варіант визначає різні значення параметрів maxretry (максимальна кількість невдалих спроб до блокування) та bantime (час блокування IP-адреси в секундах). Студенти повинні використовувати значення параметрів відповідно до свого варіанту при виконанні налаштування Fail2Ban у файлі /etc/fail2ban/jail.local. Це дозволить спостерігати і аналізувати різницю у поведінці системи захисту залежно від конфігурації параметрів.

Таблиця 3.1 – Варіанти індивідуальних завдань

№ варіанту	**maxretry**	**bantime (сек)**
1	3	300
2	4	400
3	5	500
4	6	600
5	7	700
6	8	800
7	3	900
8	4	1000
9	5	1100
10	6	1200
11	7	1300
12	8	1400
13	3	1500
14	4	1600
15	5	1700
16	6	1800
17	7	1900
18	8	2000
19	3	2100
20	4	2200
21	5	2300
22	6	2400
23	7	2500
24	8	2600
25	3	2700
26	4	2800
27	5	2900
28	6	3000
29	7	3500
30	8	4000

2 КЛЮЧОВІ ПИТАННЯ

1. У чому полягає суть атаки методом грубої сили (brute-force) на SSH-сервер, і які фактори впливають на її успішність?
2. Яку роль відіграє система журналізації rsyslog у механізмі захисту, реалізованому за допомогою Fail2Ban?
3. Як Fail2Ban виявляє підозрілу активність, і які компоненти (фільтри, дії, «в'язниці») беруть участь у цьому процесі?
4. Чому для налаштування Fail2Ban рекомендується створювати файл jail.local замість редагування jail.conf?
5. Як параметри maxretry, bantime та findtime впливають на ефективність захисту та можливість блокування легітимних користувачів?
6. Яким чином Fail2Ban взаємодіє з фаєрволом iptables для блокування IP-адрес, і як можна перевірити наявність таких правил?
7. Як правильно протестувати роботу Fail2Ban, не порушуючи стабільності системи, і які команди використовуються для імітації атаки?
8. Як аналізувати журнали Fail2Ban (/var/log/fail2ban.log) для підтвердження спрацьовування механізму блокування?
9. Чи може Fail2Ban захищати не лише SSH, а й інші сервіси (наприклад, веб-сервери або поштові служби)? Як це реалізується?
10. Які обмеження має Fail2Ban як система захисту, і які додаткові заходи (наприклад, двофакторна автентифікація, використання ключів SSH) необхідно застосовувати для комплексного захисту сервера?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Встановлення та налаштування системи журналізації rsyslog. Система *rsyslog* є ключовим компонентом безпеки Linux-сервера, оскільки саме вона забезпечує централізоване зберігання подій автентифікації, помилок служб, спроб входу та інших важливих записів. Без правильно налаштованого журналювання неможливо провести аналіз інцидентів або виявити ознаки атаки методом грубої сили. У Debian 12 пакет *rsyslog* часто встановлений за замовчуванням, але для гарантії коректної роботи рекомендується перевстановити його та налаштувати окремий файл для подій автентифікації, що значно полегшує подальший аналіз. Цей крок передбачає встановлення служби, її активацію та створення спеціалізованого конфігураційного файлу для SSH-подій.

```
sudo apt update
sudo apt install rsyslog
```

Після встановлення слід активувати службу та запустити її:

```
sudo systemctl enable --now rsyslog
sudo systemctl status rsyslog
```

Далі створюється спеціальний конфігураційний файл, який гарантує, що всі події автентифікації будуть записуватися у файл */var/log/auth.log:*

```
sudo nano /etc/rsyslog.d/10-sshd.conf
```

У відкритий файл додається наступний рядок:

```
auth,authpriv.*    /var/log/auth.log
```

Після збереження файлу служба *rsyslog* потребує перезавантаження, щоб застосувати нові налаштування:

```
sudo systemctl restart rsyslog
```

Для перевірки коректності роботи можна відстежувати події SSH у реальному часі:

```
sudo tail –f /var/log/auth.log | grep sshd
```

Якщо в терміналі з’являються записи після спроб входу (навіть локальних), це свідчить про те, що система журналювання працює коректно.

3.2 Встановлення та базова конфігурація Fail2Ban. Fail2Ban — це система запобігання вторгнень, яка аналізує журнали системних служб і автоматично блокує IP-адреси, з яких надходить підозріла активність (наприклад, множинні невдалі спроби входу через SSH). Для ефективної роботи Fail2Ban повинен бути встановлений, активований та налаштований на моніторинг правильного файлу журналу (*/var/log/auth.log*) з відповідними параметрами блокування. У цьому кроці студент встановлює пакет, активує службу та створює базову конфігурацію для захисту SSH-сервера.

```
sudo apt install fail2ban
```

Після встановлення слід увімкнути службу та переконатися, що вона працює:

```
sudo systemctl enable --now fail2ban
sudo systemctl status fail2ban
```

Далі створюється спеціальний конфігураційний файл *jail.local*, який має пріоритет над стандартними налаштуваннями і не буде перезаписаний під час оновлення пакета:

```
sudo nano /etc/fail2ban/jail.local
```

У файл додається наступний вміст (значення maxretry та bantime потрібно замінити згідно з варіантом студента):

```
[sshd]
enabled = true
port = ssh
filter = sshd
logpath = /var/log/auth.log
maxretry = 3
bantime = 3600
```

Після збереження файлу службу Fail2Ban необхідно перезапустити:

```
sudo systemctl restart fail2ban
```

Для перевірки стану захисту SSH виконується команда:

```
sudo fail2ban-client status sshd
```

У виводі має відображатися статус «enabled», шлях до файлу журналу та список заблокованих IP-адрес (на початку — порожній).

3.3 Тестування системи захисту шляхом імітації атаки. Щоб переконатися, що Fail2Ban працює коректно, необхідно імітувати атаку методом грубої сили — здійснити кілька невдалих спроб входу через SSH. Це дозволяє перевірити, чи система виявляє підозрілу активність, чи блокує IP-джерело та чи створює відповідні правила в iptables. Оскільки в лабораторному середовищі клієнт і сервер знаходяться на одній машині, імітація виконується локально через *ssh localhost*.

Для імітації атаки виконайте команду:

```
ssh fakeuser@localhost
```

У відповідь на запит пароля тричі введіть неправильне значення (наприклад, wrongpass). Після третьої невдалої спроби з’єднання має бути розірвано.

Після цього перевіряється, чи IP-адреса 127.0.0.1 (локальний хост) була додана до списку заблокованих:

```
sudo fail2ban-client status sshd
```

Також можна переглянути правила iptables, щоб переконатися, що створено правило DROP для цієї IP-адреси:

```
sudo iptables -L -n | grep DROP
```

У виводі має з’явитися рядок, що містить 127.0.0.1 у ланцюжку f2b-sshd.

3.4. Аналіз журналів роботи Fail2Ban. Для глибшого розуміння механізму роботи системи захисту необхідно проаналізувати її власні журнали. Файл */var/log/fail2ban.log* містить детальну інформацію про всі події: виявлення підозрілої активності, спрацьовування фільтрів, додавання IP-адрес до чорного списку, створення правил *iptables* тощо. Цей аналіз дозволяє підтвердити коректність налаштувань та зрозуміти логіку роботи системи.

Для перегляду останніх записів у журналі Fail2Ban виконайте:

```
sudo tail –f /var/log/fail2ban.log
```

У виводі слід знайти записи, що містять такі ключові слова:

- Found ... at ... — виявлено невдалу спробу входу;
- Ban ... — IP-адреса заблокована;
- Unban ... — IP-адреса розблокована після закінчення bantime.

Ці записи підтверджують, що система не лише реагує на атаку, але й автоматично скасовує блокування після завершення встановленого часу, що є важливою характеристикою надійного механізму захисту.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані

результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 4

Тема: Налаштування моделей дискреційного та рольового контролю доступу у Linux

Мета роботи: Ознайомитися з механізмами дискреційного контролю доступу в Linux та набути практичних навичок розмежування прав користувачів і привілеїв відповідно до ролей.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Дискреційний контроль доступу в Linux

Забезпечення безпеки інформаційних систем нерозривно пов’язане з ефективним управлінням доступом до ресурсів операційної системи. У багатокористувацьких середовищах, таких як Linux, критично важливо мати чіткі механізми, які визначають, хто, до чого і в якому обсязі має доступ. Без належного контролю доступу система стає вразливою до несанкціонованого втручання, витоку конфіденційної інформації та порушення цілісності даних. Історично операційні системи сімейства Unix, до яких належить і Linux, розроблялися з урахуванням принципів багатокористувацького доступу та ізоляції процесів, що зумовило появу потужних механізмів контролю доступу вже на рівні архітектури системи.

Основою безпеки в Linux є принцип найменших привілеїв, який передбачає надання користувачам та процесам лише тих прав, які абсолютно необхідні для виконання їхніх функцій. Цей підхід мінімізує потенційні наслідки компрометації облікового запису або програмного забезпечення, адже зловмисник отримує доступ лише до обмеженого набору ресурсів. Реалізація цього принципу вимагає ретельного планування архітектури системи, продуманого розподілу користувачів за ролями та коректного налаштування прав доступу на рівні файлової системи. Важливо розуміти, що безпека системи визначається не лише технічними засобами, а й організаційними процедурами, які регламентують створення облікових записів, призначення привілеїв та аудит дій користувачів.

Дискреційний контроль доступу, який позначається абревіатурою DAC (Discretionary Access Control), являє собою модель безпеки, в якій власник ресурсу має повноваження визначати права доступу до нього для інших користувачів. У системах Linux цей механізм реалізується через систему прав доступу до файлів та каталогів, що базується на концепції власника, групи та інших користувачів. Кожен файл або каталог у системі має асоційованого з ним власника (користувача) та групу, а також набір прав, які визначають можливість читання, запису та виконання для трьох категорій суб’єктів: власника ресурсу, членів групи-власника та всіх інших користувачів системи. Ця модель є дискреційною саме тому, що рішення про надання або обмеження

доступу приймається на розсуд власника об’єкта, а не визначається централізовано системним адміністратором.

Система прав доступу в Linux використовує дев’ятибітову структуру дозволів, яка складається з трьох триплетів по три біти кожен. Перший триплет визначає права власника файлу, другий — права групи-власника, третій — права всіх інших користувачів. Кожен біт у триплеті відповідає за конкретний тип доступу: перший біт контролює можливість читання (read), другий — запису (write), третій — виконання (execute) для файлів або переходу (traverse) для каталогів. У символьному представленні ці права позначаються літерами r, w, x відповідно, а відсутність права відображається символом дефіса. В октальному представленні, яке часто використовується в командах керування правами, кожен триплет кодується цифрою від нуля до семи, де значення обчислюється як сума: чотири для читання, два для запису, один для виконання. Наприклад, права 755 означають повний доступ для власника (7=4+2+1), читання та виконання для групи (5=4+1) і аналогічні права для інших користувачів.

Для керування правами доступу в Linux передбачено спеціалізовані команди, основними з яких є chmod для зміни режиму доступу, chown для зміни власника та chgrp для зміни групи-власника файлу. Команда chmod може працювати як з символьним, так і з октальним представленням прав, що надає адміністратору гнучкість у налаштуванні системи. При символьному використанні chmod дозволяє додавати, видаляти або встановлювати права за допомогою операторів плюс, мінус та дорівнює, а також специфікаторів u (user), g (group), o (others) та a (all). Октальний метод є компактнішим і часто використовується в скриптах автоматизації, оскільки дозволяє в одній команді встановити всі дев’ять бітів прав доступу одночасно.

Таблиця 3.1 наводить основні команди дискреційного контролю доступу, які використовуються для управління власниками, групами та правами доступу до файлів та каталогів у системі Linux.

Таблиця 3.1 — Основні команди дискреційного контролю доступу

Команда	Синтаксис	Призначення
chmod	chmod [options] mode file	Зміна прав доступу до файлу або каталогу
chown	chown [options] owner[:group] file	Зміна власника файлу або каталогу
chgrp	chgrp [options] group file	Зміна групи-власника файлу або каталогу
ls -l	ls -l [path]	Перегляд детальної інформації про файли, включно з правами доступу
umask	umask [mask]	Встановлення або перегляд маски створення файлів
getfacl	getfacl file	Отримання списків контролю доступу (ACL)
setfacl	setfacl [options] file	Встановлення списків контролю доступу (ACL)

Окрім стандартних прав доступу, Linux підтримує розширені атрибути файлів та списки контролю доступу (ACL — Access Control Lists), які надають більш гнучкі можливості керування правами. Стандартна модель DAC обмежується трьома категоріями суб'єктів, що може бути недостатнім для складних організаційних структур, де потрібно надати різні права доступу різним користувачам поза межами власника та групи. ACL дозволяють визначати права для довільної кількості користувачів та груп, створюючи детальніші політики доступу. Крім того, існують спеціальні біти прав доступу, такі як SUID (Set User ID), SGID (Set Group ID) та sticky bit, які модифікують поведінку системи при виконанні файлів або роботі з каталогами, надаючи додаткові можливості для реалізації складних схем безпеки.

1.2 Управління користувачами та групами

Ефективна реалізація дискреційного контролю доступу неможлива без продуманої організації користувачів та груп у системі. В Linux кожен користувач ідентифікується унікальним числовим ідентифікатором UID (User Identifier) та належить принаймні до однієї групи, яка також має числовий ідентифікатор GID (Group Identifier). Інформація про користувачів зберігається в текстовому файлі /etc/passwd, який містить рядки з сімома полями, розділеними двокрапками: ім'я користувача, символ x (що вказує на зберігання паролю в окремому файлі), UID, GID основної групи, поле коментаря, шлях до домашнього каталогу та командну оболонку. Паролі користувачів зберігаються в зашифрованому вигляді у файлі /etc/shadow, який доступний лише для читання суперкористувачу root, що запобігає несанкціонованому доступу до хешів паролів навіть при компрометації основного файлу облікових записів.

Групи в Linux призначені для об'єднання користувачів з подібними повноваженнями або функціональними обов'язками, що спрощує керування правами доступу до спільних ресурсів. Інформація про групи зберігається у файлі /etc/group, кожен рядок якого містить чотири поля: назву групи, символ x (для сумісності з механізмом групових паролів), GID та список членів групи через кому. Кожен користувач має одну первинну (primary) групу, яка вказана в його записі в /etc/passwd, та може належати до довільної кількості додаткових (supplementary) груп, перелік яких міститься в /etc/group. Первинна група автоматично призначається власником групи для всіх файлів, які створює користувач, якщо не встановлено спеціальний біт SGID на каталозі, що змінює цю поведінку.

Створення та управління обліковими записами в Linux виконується за допомогою спеціалізованих команд, які забезпечують не лише додавання записів до конфігураційних файлів, а й виконання супутніх операцій, таких як створення домашнього каталогу, копіювання шаблонних конфігураційних файлів та ініціалізація поштової скриньки. Команда useradd надає низькорівневий інтерфейс для створення користувача з детальним контролем всіх параметрів, тоді як adduser (доступна в дистрибутивах на базі Debian) є

інтерактивним скриптом високого рівня, який запитує додаткову інформацію та автоматично виконує типові налаштування. Аналогічно, команда groupadd створює нову групу в системі, а usermod дозволяє модифікувати параметри існуючого облікового запису, включно з додаванням користувача до додаткових груп за допомогою опції -aG, яка зберігає членство в інших групах.

Таблиця 3.2 демонструє команди для управління користувачами та групами, які необхідні для організації рольової структури доступу в багатокористувацькій системі.

Таблиця 3.2 — Команди управління користувачами та групами

Команда	Синтаксис	Призначення
useradd	useradd [options] username	Створення нового користувача (низькорівнева)
adduser	adduser [options] username	Створення нового користувача (високорівнева, інтерактивна)
userdel	userdel [options] username	Видалення користувача з системи
usermod	usermod [options] username	Модифікація параметрів існуючого користувача
groupadd	groupadd [options] groupname	Створення нової групи
groupdel	groupdel groupname	Видалення групи з системи
groupmod	groupmod [options] groupname	Модифікація параметрів існуючої групи
passwd	passwd [username]	Зміна пароля користувача
getent	getent database key	Отримання записів з системних баз даних
id	id [username]	Відображення UID, GID та груп користувача
groups	groups [username]	Відображення груп, до яких належить користувач

Важливим аспектом безпеки при управлінні користувачами є принцип розділення обов’язків та створення окремих облікових записів для різних функціональних ролей. Використання спільних облікових записів або надання всім користувачам адміністративних привілеїв суттєво ускладнює аудит дій, унеможливлює персональну відповідальність за дії в системі та збільшує ризики безпеки. Кожен користувач повинен мати унікальний обліковий запис з чітко визначеними повноваженнями, які відповідають його посадовим обов’язкам. Групи використовуються для об’єднання користувачів з однаковими або схожими функціями, що дозволяє централізовано керувати їхнім доступом до ресурсів шляхом призначення прав на рівні групи, а не окремих користувачів.

1.3 Механізм sudo та рольове розмежування привілеїв

У традиційних Unix-подібних системах адміністративні операції вимагають повного переключення на обліковий запис суперкористувача root, що створює ризики безпеки через надмірні привілеї та ускладнює аудит дій. Механізм sudo (substitute user do) пропонує альтернативний підхід, дозволяючи звичайним користувачам виконувати окремі команди з підвищеними привілеями без необхідності знати пароль root та без отримання повного доступу до системи. Цей інструмент реалізує концепцію рольового розмежування доступу (RBAC — Role-Based Access Control), де права надаються на основі функціональних ролей, а не безпосередньо користувачам. Використання sudo дозволяє створювати детальні політики доступу, в яких точно визначається, хто, коли, на яких хостах та які команди може виконувати з привілеями іншого користувача або root.

Конфігурація *sudo* зберігається в спеціальному файлі */etc/sudoers*, який має критичне значення для безпеки системи і тому ніколи не редагується безпосередньо текстовими редакторами. Для внесення змін використовується команда *visudo*, яка відкриває файл в редакторі з автоматичною перевіркою синтаксису перед збереженням. Якщо в конфігурації виявлено помилки, *visudo* не дозволить зберегти файл, що запобігає ситуації, коли через синтаксичну помилку система втрачає можливість виконання адміністративних команд. Синтаксис файлу sudoers підтримує різноманітні директиви, але основною є специфікація правил доступу, яка має формат: користувач або група, хости, від імені якого користувача можуть виконуватися команди, та список дозволених команд з їхніми аргументами.

Правила в файлі *sudoers* мають структуру, яка складається з чотирьох основних компонентів, розділених пробілами та спеціальними символами. Перший компонент визначає користувача або групу (група вказується з префіксом відсотка), другий — список хостів, на яких застосовується правило (зазвичай ALL для всіх хостів), третій — користувача, від імені якого виконуватиметься команда (у дужках, часто ALL або root), четвертий — список дозволених команд з повними шляхами до виконуваних файлів. Наприклад, правило %operators ALL=(ALL) /bin/systemctl status дозволяє всім членам групи operators на будь-якому хості виконувати команду /bin/systemctl status від імені будь-якого користувача, але лише з аргументом status, що обмежує можливість зміни стану системних служб. Можна також використовувати модифікатори, такі як NOPASSWD для виконання команд без запиту пароля або специфікації псевдонімів для групування користувачів, хостів або команд.

Таблиця 3.3 містить приклади конфігурацій sudo для різних сценаріїв рольового доступу, які демонструють гнучкість механізму делегування привілеїв.

Таблиця 3.3 — Приклади правил конфігурації sudo

Правило	Опис
%operators ALL=(ALL) /bin/systemctl status	Група operators може переглядати стан служб
%developers ALL=(ALL) /usr/bin/git, /usr/bin/docker	Група developers може використовувати git та docker
admin_user ALL=(ALL) NOPASSWD: /usr/sbin/service	Користувач admin_user керує службами без пароля
%backup ALL=(root) /usr/bin/rsync, /bin/tar	Група backup виконує команди резервного копіювання від імені root
monitoring_user ALL=(ALL) NOPASSWD: /usr/bin/systemctl status *	Користувач monitoring_user переглядає стан всіх служб без пароля
%webadmins ALL=(www-data) /usr/sbin/apache2ctl	Група webadmins керує Apache від імені користувача www-data

Ефективне використання sudo вимагає ретельного планування рольової структури організації та документування правил доступу. Занадто широкі правила, такі як надання повного доступу до всіх команд або використання підстановочних знаків без обмежень, нівелюють переваги механізму та створюють ризики безпеки, порівнянні з прямим доступом до облікового запису *root*. Правила повинні бути максимально специфічними, обмежуючи не лише команди, а й їхні аргументи, де це можливо. Важливо враховувати, що деякі команди можуть бути використані для отримання shell-доступу або виконання довільних команд (наприклад, текстові редактори, інтерпретатори скриптів), тому їх делегування вимагає особливої обережності. Регулярний аудит файлу sudoers та логів використання sudo (які зберігаються в системному журналі) є необхідною практикою для виявлення надлишкових привілеїв або спроб несанкціонованого доступу.

1.4 Filesystem Hierarchy Standard та організація каталогів безпеки

Стандарт ієрархії файлової системи (FHS — Filesystem Hierarchy Standard) визначає структуру каталогів в Unix-подібних операційних системах, встановлюючи призначення основних директорій та правила розміщення файлів. Дотримання цього стандарту забезпечує передбачуваність розташування системних ресурсів, спрощує адміністрування, підвищує сумісність між різними дистрибутивами Linux та полегшує створення переносимих скриптів і програм. У контексті безпеки особливе значення мають каталоги, які містять конфігураційні файли, дані служб та тимчасові файли, оскільки саме вони часто стають об’єктами атак або потребують спеціальних налаштувань прав доступу.

Каталог */srv* призначений для зберігання даних, що надаються специфічними службами системи, такими як веб-сервери, FTP-сервери або файлові сховища. На відміну від каталогу */var*, який містить змінні дані загального призначення, */srv* використовується для структурованого

розміщення даних конкретних сервісів, де адміністратор може створювати підкаталоги відповідно до організаційних потреб. Такий підхід дозволяє легко ідентифікувати дані різних служб та застосовувати до них диференційовані політики резервного копіювання та контролю доступу. Конфігураційні файли служб зазвичай розміщуються в каталозі */etc*, але дані, з якими ці служби працюють, рекомендується зберігати саме в */srv*, що відокремлює конфігурацію від даних та спрощує міграцію або відновлення системи.

При проектуванні структури каталогів для захищеного зберігання даних важливо враховувати принципи розділення за рівнями конфіденційності та функціональним призначенням. Каталоги з конфіденційною інформацією повинні мати обмежені права доступу на рівні файлової системи, встановлені таким чином, щоб лише авторизовані користувачі або процеси могли отримати до них доступ. Типова практика передбачає встановлення власником каталогу користувача root або спеціалізованого системного користувача, призначення групою — функціональної групи, що об’єднує користувачів з відповідними повноваженнями, та налаштування прав у форматі 750 або 770 для каталогів і 640 або 660 для файлів. Такі налаштування забезпечують, що лише власник має повний контроль, члени групи можуть читати або виконувати необхідні операції, а всі інші користувачі системи повністю ізольовані від ресурсу.

Рисунок 3.1 ілюструє типову ієрархію каталогів для організації захищених даних служби в каталозі */srv/secure_data* з відповідними правами доступу та структурою власності.

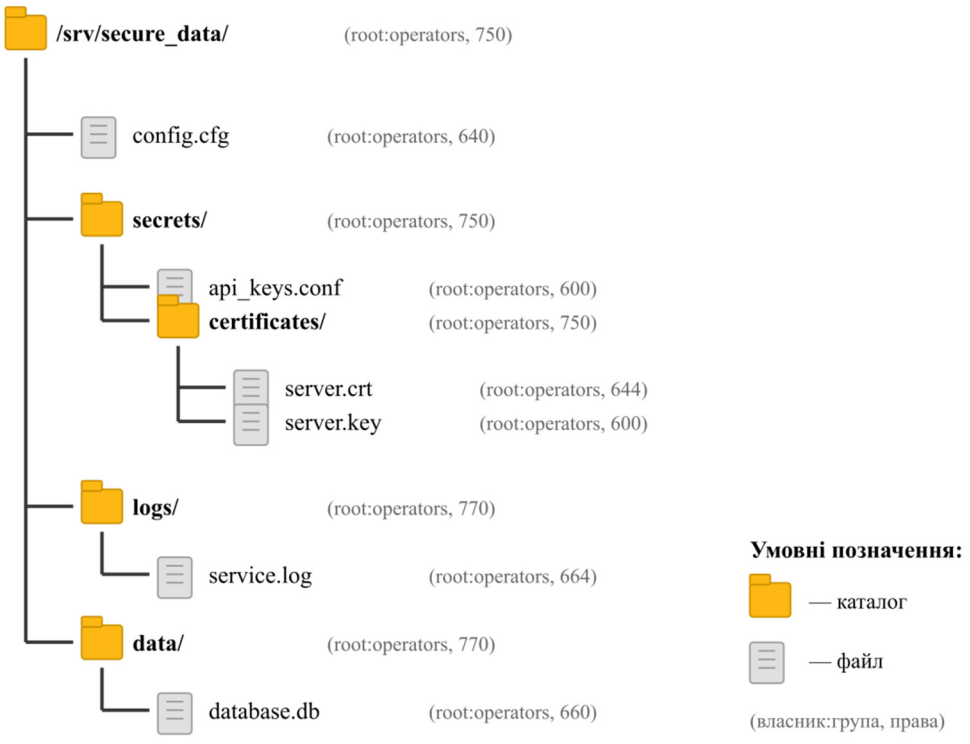

Рисунок 3.1 — Ієрархія каталогів для захищених даних

На рисунку 3.1 показано структуру каталогів, де кореневий каталог /srv/secure_data належить користувачу root та групі operators з правами 750, що

дозволяє власнику повний доступ, групі — читання та перехід, а іншим користувачам доступ заборонено. Всередині розміщено конфігураційний файл *config.cfg* з правами 640, доступний для читання групі *operators*. Підкаталог secrets містить конфіденційні дані, такі як API-ключі з більш обмеженими правами 600 (доступ лише власнику) та сертифікати безпеки з диференційованими правами залежно від їх чутливості. Каталог logs має права 770, що дозволяє як власнику, так і групі повний доступ для запису логів, а каталог data також відкритий для запису групі для роботи з базами даних або іншими файлами даних.

1.5 Практичні аспекти тестування та аудиту контролю доступу

Налаштування механізмів контролю доступу вимагає обов'язкової верифікації коректності конфігурації через систематичне тестування з різних облікових записів. Процес тестування включає імітацію дій користувачів з різними рівнями привілеїв та перевірку відповідності результатів очікуваній поведінці системи. Для цього використовується команда su (substitute user) з опцією дефіса, яка забезпечує повне переключення на середовище цільового користувача, включно з його домашнім каталогом, змінними оточення та оболонкою. Альтернативно, команда *sudo -u username -i* дозволяє адміністратору виконувати команди від імені іншого користувача, що корисно для автоматизованого тестування без необхідності знати паролі облікових записів.

Методологія тестування контролю доступу передбачає створення тестових сценаріїв, які покривають всі можливі комбінації користувачів, ресурсів та операцій. Для кожної ролі необхідно перевірити як дозволені операції (позитивне тестування), так і заборонені дії (негативне тестування), щоб переконатися, що система правильно блокує несанкціонований доступ. Типові тестові випадки включають спроби читання файлів без належних прав, виконання команд через sudo поза визначеними в конфігурації, створення або модифікацію файлів у захищених каталогах та спроби підвищення привілеїв через використання вразливих команд. Результати тестування документуються з фіксацією отриманих повідомлень про помилки, кодів повернення команд та змін стану системи.

Аудит системи контролю доступу є невід'ємною частиною забезпечення безпеки та включає регулярний аналіз конфігураційних файлів, журналів подій та фактичного стану прав доступу до файлів і каталогів. Система Linux веде детальні журнали використання sudo через механізм *syslog* або *systemd*-журнал, записуючи кожну спробу виконання команди, успішну чи невдалу, з інформацією про користувача, термінал, робочий каталог та саму команду. Ці журнали аналізуються на предмет підозрілої активності, такої як численні невдалі спроби виконання заборонених команд, використання привілейованих команд у незвичний час або з незвичних терміналів, або спроби виконання команд користувачами, які не повинні мати відповідних привілеїв. Інструменти

автоматизованого аналізу логів можуть виявляти аномалії та сповіщати адміністраторів про потенційні інциденти безпеки в реальному часі.

Таблиця 3.4 представляє команди для тестування та аудиту налаштувань контролю доступу, які використовуються для перевірки коректності конфігурації та виявлення проблем безпеки.

Таблиця 3.4 — Команди тестування та аудиту контролю доступу

Команда	Синтаксис	Призначення
su	su - username	Переключення на іншого користувача з його середовищем
sudo -l	sudo -l [-U username]	Перегляд дозволених sudo-команд для користувача
visudo -c	visudo -c	Перевірка синтаксису файлу sudoers
journalctl	journalctl -u sudo	Перегляд журналу подій використання sudo
find	find /path -perm mode	Пошук файлів з певними правами доступу
namei	namei -l /path/to/file	Відображення прав доступу для всього шляху до файлу
last	last [username]	Перегляд історії входів користувачів
aureport	aureport --auth	Звіт про події аутентифікації (при увімкненому auditd)

Регулярний аудит також повинен включати перевірку відповідності фактичних налаштувань прав доступу задокументованим політикам безпеки організації. Використання автоматизованих інструментів для сканування файлової системи дозволяє виявляти файли з надмірно широкими правами, некоректними власниками або аномальними атрибутами, що можуть вказувати на компрометацію системи або помилки конфігурації. Результати аудиту повинні документуватися та використовуватися для постійного вдосконалення політик безпеки та процедур управління доступом, забезпечуючи адаптацію системи захисту до змін в організаційній структурі та загрозах безпеці.

2 КЛЮЧОВІ ПИТАННЯ

1. У чому полягає сутність дискреційного контролю доступу (DAC) і чим він відрізняється від мандатного контролю доступу (MAC)? Яку роль відіграє власник ресурсу в моделі DAC?

2. Яку структуру має дев'ятибітова система прав доступу в Linux? Поясніть значення кожного з трьох триплетів та різницю між символьним і октальним представленням прав доступу.

3. Які команди використовуються для управління правами доступу, власниками та групами файлів? Наведіть приклади використання команд chmod, chown та chgrp з різними опціями.

4. Яке призначення файлів /etc/passwd, /etc/shadow та /etc/group у системі Linux? Опишіть структуру записів у цих файлах та їх роль у механізмі аутентифікації.

5. У чому різниця між первинною (primary) та додатковими (supplementary) групами користувача? Як належність до групи впливає на права доступу до файлів та каталогів?

6. Що таке механізм sudo і як він реалізує принцип найменших привілеїв? Чому необхідно використовувати команду visudo для редагування конфігурації sudo?

7. Опишіть структуру правила в файлі /etc/sudoers. Що означає правило %operators ALL=(ALL) /bin/systemctl status і які обмеження воно накладає?

8. Яке призначення каталогу /srv згідно зі стандартом FHS? Чому рекомендується зберігати конфіденційні дані служб саме в цьому каталозі, а не в інших системних директоріях?

9. Які права доступу (у форматі rwx та октальному) слід встановити для каталогу з конфіденційними даними, якщо доступ повинен мати лише власник (повний) та група (читання і перехід)? Обґрунтуйте вибір.

10. Які методи та інструменти використовуються для тестування та аудиту налаштувань контролю доступу? Яку інформацію можна отримати з журналів використання sudo та чому це важливо для безпеки системи?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1. Створення користувачів та груп згідно з функціональними ролями. У багатокористувацьких системах Linux важливо чітко розмежовувати облікові записи за функціональними ролями, що дозволяє реалізувати принцип найменших привілеїв. Для цього створюються окремі користувачі (admin_user, operator_user, regular_user) та група operators, яка об’єднує операторів із подібними повноваженнями. Команда adduser автоматично створює домашній каталог, додає запис до /etc/passwd та ініціалізує оболонку, а команда groupadd — нову групу. Після цього користувача operator_user додають до групи operators за допомогою usermod -aG, що гарантує збереження його належності до інших груп. Перевірка через getent підтверджує, що облікові записи та група створені коректно.

```
sudo adduser admin_user
sudo adduser operator_user
sudo adduser regular_user
sudo groupadd operators
sudo usermod –aG operators operator_user
getent passwd | grep –E 'admin_user|operator_user|regular_user'
getent group operators
```

3.2. Налаштування каталогу та файлів для захищеного доступу. Для моделювання службового середовища створюється каталог /srv/secure_data — стандартне місце для даних, що використовуються службами (за рекомендаціями Filesystem Hierarchy Standard). У ньому розміщується конфігураційний файл config.cfg, який буде доступний лише авторизованим особам. Власником каталогу призначається користувач root, а групою — operators, що дозволяє групі отримувати спільний доступ. Права 750 означають: повний доступ для власника, читання та виконання (перехід) для групи, і жодного доступу для інших. Аналогічно, права 640 на файл забезпечують читання/запис власнику та лише читання членам групи. Такий підхід запобігає несанкціонованому доступу з боку звичайних користувачів.

```
sudo mkdir /srv/secure_data
sudo touch /srv/secure_data/config.cfg
sudo chown root:operators /srv/secure_data
sudo chmod 750 /srv/secure_data
sudo chmod 640 /srv/secure_data/config.cfg
ls -ld /srv/secure_data /srv/secure_data/config.cfg
```

3.3. Налаштування рольового доступу через sudo. Механізм sudo дозволяє делегувати обмежені адміністративні повноваження без надання повного доступу до системи. Для цього використовується спеціальний файл /etc/sudoers, який ніколи не редагується напряму, а лише через команду visudo. Ця команда забезпечує синтаксичну перевірку перед збереженням, що запобігає блокуванню системи через помилки. У нашому випадку правило %operators ALL=(ALL) /bin/systemctl status дозволяє всім членам групи operators переглядати стан будь-якого системного сервісу, але не змінювати його. Це типовий приклад безпечного делегування: оператор може моніторити систему, але не втручатися в її роботу. Перевірка синтаксису через visudo -c є обов’язковим етапом після внесення змін.

```
sudo visudo
# Додати рядок:
%operators ALL=(ALL) /bin/systemctl status
sudo visudo -c
```

3.4. Тестування механізмів контролю доступу. Перевірка прав доступу проводиться шляхом імітації дій різних користувачів. Для цього використовується команда su - username, яка повністю імітує сесію користувача (з його оболонкою, змінними середовища тощо). Від імені operator_user має бути доступ до каталогу /srv/secure_data та можливість виконання sudo systemctl status ssh, але заборонено перезапускати сервіс. Користувач regular_user не має доступу до каталогу і не може виконувати sudo-команди, що підтверджує ефективність DAC та налаштованих прав. Адміністратор admin_user не має спеціальних прав у sudoers, тому його можливості обмежені стандартними привілеями, що демонструє важливість явного налаштування ролей.

```
# Приклад для operator_user:
su - operator_user
ls /srv/secure_data
sudo systemctl status ssh
sudo systemctl restart ssh  # має відмовити
exit
```

3.5. Аналіз результатів. Після тестування формується аналіз: які операції були дозволені, а які — заблоковані, і чи відповідає це очікуваному поведінку. Наприклад, якщо regular_user отримав доступ до config.cfg, це свідчить про помилку у налаштуванні прав DAC. Якщо operator_user зміг перезапустити сервіс, це означає, що правило в sudoers занадто широке. Такий аналіз дозволяє зрозуміти, як механізми дискреційного контролю доступу (DAC) та рольового розмежування (через sudo) разом реалізують принцип найменших привілеїв — ключовий підхід до безпеки в сучасних операційних системах.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 5

Тема: Моделювання порушень CIA-триади та механізмів захисту в Linux

Мета роботи: Сформувати практичне розуміння властивостей CIA-триади та продемонструвати типові сценарії їх порушення і базові засоби захисту в середовищі Linux.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Вступ до фундаментальних властивостей інформаційної безпеки

Інформаційна безпека як науково-прикладна дисципліна базується на фундаментальній концепції, відомій як CIA-триада, що представляє три основоположні властивості захищених інформаційних систем: конфіденційність (Confidentiality), цілісність (Integrity) та доступність (Availability). Ця модель, вперше сформульована у 1970-х роках та офіційно закріплена у стандартах ISO/IEC 27000, визначає мінімально необхідний набір характеристик, які повинна забезпечувати будь-яка система для гарантування захисту інформаційних активів. Кожна з трьох властивостей відповідає за окремий аспект безпеки: конфіденційність гарантує, що інформація доступна лише авторизованим суб'єктам, цілісність забезпечує відсутність несанкціонованих модифікацій, а доступність підтримує можливість авторизованого доступу до ресурсів у потрібний момент часу. Важливо розуміти, що ці три властивості не є ізольованими компонентами, а формують взаємопов'язану систему, де порушення однієї властивості часто призводить до компрометації інших, створюючи каскадний ефект у безпеці всієї інформаційної інфраструктури.

Практичне значення CIA-триади полягає у тому, що вона надає структурований підхід до аналізу загроз та проектування систем захисту інформації. При розробці архітектури безпеки кожен компонент системи оцінюється з точки зору того, які з трьох властивостей він має забезпечувати та яким загрозам він може піддаватися. Наприклад, система аутентифікації первинно спрямована на забезпечення конфіденційності через контроль доступу, але водночас повинна підтримувати високу доступність, щоб не блокувати легітимних користувачів, та гарантувати цілісність облікових записів для запобігання підміні ідентифікаційних даних. Така багатовимірність вимагає від фахівців з інформаційної безпеки комплексного підходу, де кожне рішення аналізується з позиції впливу на всі три властивості одночасно, враховуючи можливі компроміси між ними.

1.2 Конфіденційність як властивість інформаційної системи

Конфіденційність визначається як властивість інформаційної системи, що гарантує недоступність інформації для несанкціонованих суб'єктів, включаючи як людей, так і програмні компоненти. Ця властивість реалізується через комплекс технічних і організаційних заходів, серед яких ключову роль відіграють механізми контролю доступу, криптографічний захист та політики безпеки. Порушення конфіденційності може відбуватися на різних рівнях інформаційної системи: від фізичного доступу до носіїв інформації до логічного перехоплення мережевого трафіку або експлуатації вразливостей у програмному забезпеченні. Наслідки таких порушень варіюються залежно від типу скомпрометованої інформації та можуть включати фінансові втрати, репутаційні ризики, порушення нормативних вимог або навіть загрозу національній безпеці у випадку державних систем.

Криптографія є основним технічним інструментом забезпечення конфіденційності у сучасних інформаційних системах. Процес шифрування перетворює початковий текст (plaintext) у шифротекст (ciphertext) за допомогою криптографічного алгоритму та ключа, роблячи інформацію нечитабельною для всіх, хто не володіє відповідним ключем розшифрування. Розрізняють симетричне шифрування, де один і той самий ключ використовується для шифрування та розшифрування (наприклад, AES-256), та асиметричне шифрування з парою ключів – відкритим та закритим (наприклад, RSA або ECC). Вибір конкретного криптографічного алгоритму залежить від контексту застосування: симетричні алгоритми забезпечують вищу швидкість обробки і використовуються для шифрування великих обсягів даних, тоді як асиметричні алгоритми зазвичай застосовуються для безпечного обміну симетричними ключами та створення цифрових підписів.

Захист даних під час передачі мережею, відомий як захист "in transit", є критично важливим аспектом забезпечення конфіденційності у розподілених системах. Незахищені протоколи, такі як HTTP, FTP або Telnet, передають інформацію у відкритому вигляді, що робить її вразливою до атак типу "man-in-the-middle" або пасивного прослуховування мережевого трафіку. Для захисту використовуються криптографічні протоколи транспортного рівня, найбільш поширеними з яких є TLS (Transport Layer Security) для веб-трафіку та SSH (Secure Shell) для віддаленого адміністрування та передачі файлів. Протокол SSH, який широко використовується в Unix-подібних системах, забезпечує не лише шифрування каналу зв'язку, але й автентифікацію сторін за допомогою криптографічних ключів, що суттєво підвищує рівень безпеки порівняно з традиційними методами парольної автентифікації.

Інструменти мережевого аналізу, такі як tcpdump, Wireshark або tshark, дозволяють перехоплювати та аналізувати мережевий трафік на рівні пакетів, що робить їх незамінними як для діагностики мережевих проблем, так і для аудиту безпеки. Команда tcpdump працює в режимі командного рядка і захоплює пакети, що проходять через мережеві інтерфейси, дозволяючи фільтрувати їх за різними критеріями (порт, протокол, IP-адреса) та

відображати вміст у текстовому або шістнадцятковому форматі. При аналізі незашифрованого HTTP-трафіку опція “-A” дозволяє відобразити ASCII-вміст пакетів, де легко можна побачити передані дані, включаючи паролі, API-ключі або іншу чутливу інформацію. Натомість при аналізі SSH-трафіку на порту 22 буде видно лише зашифровані байти, які неможливо інтерпретувати без знання сесійних ключів, що наочно демонструє ефективність криптографічного захисту. Важливо зазначити, що використання таких інструментів у промислових мережах має бути санкціоноване та відповідати політикам безпеки організації, оскільки неконтрольоване перехоплення трафіку може порушувати приватність користувачів та законодавство про захист даних.

Таблиця 5.1 наводить основні команди для роботи з криптографічними протоколами та мережевим аналізом у контексті забезпечення конфіденційності.

Таблиця 5.1 – Команди для забезпечення та тестування конфіденційності

Команда	Призначення	Приклад використання
scp source user@host:dest	Безпечне копіювання файлів через SSH з шифруванням	scp secret.txt admin@192.168.1.10:/tmp/
ssh user@host	Встановлення зашифрованого з’єднання з віддаленим хостом	ssh root@server.example.com
tcpdump -i interface -A port N	Перехоплення та відображення ASCII-вмісту пакетів на порту N	tcpdump -i eth0 -A port 8080
python3 -m http.server port	Запуск простого HTTP-сервера (незахищений)	python3 -m http.server 8080
wget http://host/file	Завантаження файлу через HTTP (незахищений протокол)	wget http://192.168.1.10:8080/data.txt
openssl enc -aes-256-cbc -in file -out file.enc	Шифрування файлу алгоритмом AES-256	openssl enc -aes-256-cbc -in doc.txt -out doc.enc

Механізми контролю доступу доповнюють криптографічний захист, реалізуючи принцип найменших привілеїв та розділення обов’язків. У операційних системах сімейства Linux традиційна модель дискреційного контролю доступу (DAC) базується на правах власника, групи та інших користувачів, виражених через біти дозволів читання, запису та виконання. Більш сучасні механізми, такі як мандатний контроль доступу (MAC) у реалізаціях SELinux або AppArmor, дозволяють встановлювати детальні політики безпеки на рівні системи, обмежуючи можливості навіть привілейованих процесів. Правильна конфігурація прав доступу до конфіденційних файлів передбачає обмеження можливості читання лише для власника або вузького кола довірених користувачів, що зменшує поверхню атаки у разі компрометації системи.

1.3 Цілісність даних та механізми її забезпечення

Цілісність у контексті інформаційної безпеки визначає властивість даних зберігати свій первісний стан та залишатися незміненими під час зберігання, обробки або передачі, якщо такі зміни не були санкціоновані легітимними суб'єктами. Порушення цілісності може бути наслідком як навмисних дій зловмисників (модифікація конфігураційних файлів, впровадження шкідливого коду, підміна виконуваних файлів), так і ненавмисних факторів (апаратні збої, помилки програмного забезпечення, пошкодження носіїв інформації). Наслідки порушення цілісності часто є більш критичними, ніж порушення конфіденційності, оскільки модифіковані дані можуть призводити до прийняття неправильних рішень, виконання шкідливих операцій або повної компрометації системи, при цьому користувачі можуть навіть не підозрювати про існування проблеми.

Криптографічні хеш-функції є фундаментальним інструментом контролю цілісності та широко застосовуються у різних сценаріях інформаційної безпеки. Хеш-функція перетворює вхідні дані довільного розміру у фіксовану послідовність байтів (хеш, дайджест або контрольну суму), що володіє кількома важливими властивостями: детермінованістю (однакові дані завжди дають однаковий хеш), односпрямованістю (практично неможливо відновити початкові дані з хешу) та стійкістю до колізій (практично неможливо знайти два різних набори даних з однаковим хешем). Найбільш поширені сучасні алгоритми хешування включають SHA-256 та SHA-512 із сімейства SHA-2, які вважаються криптографічно стійкими, тоді як старіші алгоритми MD5 та SHA-1 більше не рекомендуються для використання у критичних застосуваннях через виявлені вразливості до атак на колізії.

Практичне застосування хеш-функцій для контролю цілісності передбачає обчислення хешу файлу або набору файлів на момент їх створення або першої інсталяції з подальшим збереженням цих еталонних значень у захищеному сховищі. Періодично або за запитом виконується повторне обчислення хешу поточного стану файлів та порівняння з еталонними значеннями. Будь-яка розбіжність свідчить про модифікацію файлу, що вимагає подальшого розслідування для визначення, чи була зміна легітимною (оновлення програмного забезпечення) чи результатом компрометації. Такий підхід реалізований у спеціалізованих системах виявлення втручань на базі хостів (HIDS), таких як AIDE (Advanced Intrusion Detection Environment), Tripwire або OSSEC, які автоматизують процес моніторингу цілісності критичних системних файлів, конфігурацій та виконуваних модулів.

У операційних системах Linux команда sha256sum є стандартним інструментом для обчислення SHA-256 хешів файлів. Базовий синтаксис передбачає вказівку імені файлу, після чого виводиться 64-символьний шістнадцятковий рядок хешу та ім'я файлу. Для автоматизованої перевірки цілісності використовується можливість збереження хешу у файл з подальшою верифікацією за допомогою опції "-c", яка читає збережені хеші та порівнює їх з поточними значеннями, виводячи статус OK для незмінених файлів та

FAILED для модифікованих. Важливо зберігати файли з еталонними хешами у захищеному місці, оскільки їх компрометація дозволить зловмиснику модифікувати як цільовий файл, так і відповідний хеш, роблячи систему контролю цілісності неефективною.

Таблиця 5.2 систематизує основні команди та утиліти для роботи з контролем цілісності даних у Linux-системах.

Таблиця 5.2 – Команди для забезпечення та перевірки цілісності

Команда	Призначення	Приклад використання
sha256sum file	Обчислення SHA-256 хешу файлу	sha256sum document.pdf
sha256sum file > hash.txt	Збереження хешу файлу для подальшої перевірки	sha256sum config.conf > config.sha256
sha256sum -c hash.txt	Перевірка цілісності файлу за збереженим хешем	sha256sum -c backup.sha256
md5sum file	Обчислення MD5 хешу (застарілий, не рекомендується)	md5sum legacy_file.dat
sha512sum file	Обчислення SHA-512 хешу (вища криптостійкість)	sha512sum critical_binary
gpg --sign file	Створення цифрового підпису файлу	gpg --sign contract.pdf
gpg --verify file.sig	Перевірка цифрового підпису файлу	gpg --verify contract.pdf.sig

Цифрові підписи представляють більш досконалий механізм забезпечення цілісності, який поєднує контроль недоторканості даних з автентифікацією джерела інформації. На відміну від простого хешування, цифровий підпис створюється шляхом шифрування хешу документа приватним ключем автора, що дозволяє будь-якій стороні, що володіє відповідним відкритим ключем, не лише переконатися у відсутності змін у документі, але й достовірно встановити особу його підписувача. Стандарт цифрового підпису широко використовується для забезпечення цілісності програмного забезпечення, де розробники підписують свої дистрибутиви, дозволяючи користувачам верифікувати автентичність та недоторканість завантажених пакетів. У Linux-екосистемі інструмент GnuPG (GPG) реалізує повний набір функцій для роботи з цифровими підписами та шифруванням за стандартом OpenPGP, забезпечуючи як захист конфіденційності, так і гарантії цілісності та автентичності.

Системи контролю версій, такі як Git, також використовують хеш-функції для забезпечення цілісності історії змін програмного коду. Кожен коміт у Git ідентифікується унікальним SHA-1 хешем, який обчислюється на основі вмісту всіх файлів, метаданих коміту та хешу батьківського коміту, створюючи криптографічний ланцюжок, де модифікація будь-якого історичного запису автоматично змінює хеші всіх наступних комітів, роблячи підміну історії легко виявлюваною. Такий підхід демонструє, як криптографічні примітиви можуть бути інтегровані в інструменти розробки для прозорого забезпечення цілісності без необхідності додаткових дій від користувачів.

1.4 Доступність як критична властивість інформаційних систем

Доступність визначає здатність інформаційної системи надавати ресурси та сервіси авторизованим користувачам у потрібний момент часу, забезпечуючи безперервність бізнес-процесів та виконання критичних функцій. На відміну від конфіденційності та цілісності, які фокусуються на захисті інформації від несанкціонованого доступу або модифікації, доступність орієнтована на забезпечення стабільного функціонування системи навіть в умовах несприятливих факторів, включаючи технічні відмови, надмірне навантаження або цілеспрямовані атаки. Порушення доступності може мати серйозні наслідки для організацій, особливо у сферах, де простої є неприпустимими: фінансові сервіси, медичні системи, критична інфраструктура, електронна комерція або телекомунікації.

Атаки типу відмови в обслуговуванні (Denial of Service, DoS) являють собою цілеспрямовані дії, спрямовані на порушення доступності системи шляхом вичерпання її обмежених ресурсів, таких як процесорний час, оперативна пам’ять, мережева пропускна здатність, дискові операції введення-виведення або кількість одночасних з’єднань. Розрізняють локальні DoS-атаки, що виконуються з легітимного облікового запису всередині системи, та мережеві DoS-атаки, що надходять ззовні через мережеві протоколи. Особливо небезпечними є розподілені атаки типу DDoS (Distributed Denial of Service), де зловмисник використовує мережу скомпрометованих комп’ютерів (ботнет) для генерації масивного трафіку, який може перевищувати пропускну здатність каналів зв’язку або обчислювальні можливості цільової системи на кілька порядків. Такі атаки важко відбивати традиційними методами фільтрації, оскільки кожен окремий запит може виглядати легітимним, а проблема полягає лише в їх надмірній кількості.

Механізми обмеження ресурсів на рівні операційної системи є одним із базових методів захисту від локальних DoS-атак та запобігання ситуаціям, коли один процес або користувач може повністю монополізувати системні ресурси на шкоду іншим. У Unix-подібних системах це реалізується через концепцію resource limits, яка дозволяє встановлювати м’які (soft) та жорсткі (hard) обмеження на використання різних типів ресурсів для окремих процесів, користувачів або груп. М’які обмеження можуть бути збільшені процесом до значення жорсткого обмеження, тоді як жорсткі обмеження може змінювати лише привілейований користувач. Такий підхід забезпечує гнучкість у керуванні ресурсами, дозволяючи адміністраторам встановлювати різні політики для різних категорій користувачів відповідно до їх потреб та рівня довіри.

Команда ulimit у оболонці bash надає інтерфейс для перегляду та модифікації обмежень ресурсів для поточної оболонки та її дочірніх процесів. Найбільш важливі типи обмежень включають максимальний час виконання процесу в CPU-секундах (опція “-t”), максимальний розмір віртуальної пам’яті (опція “-v”), максимальну кількість відкритих файлових дескрипторів (опція “-n”), максимальну кількість процесів користувача (опція “-u”) та максимальний

розмір файлів, які можуть бути створені (опція “-f”). Обмеження процесорного часу є особливо ефективним проти процесів, які створюють безкінечні цикли або виконують ресурсомісткі обчислення без обмежень, оскільки після досягнення ліміту ядро операційної системи автоматично надсилає процесу сигнал SIGXCPU, який за замовчуванням призводить до його завершення.

Таблиця 5.3 демонструє основні команди для управління ресурсами та захисту доступності в Linux-системах.

Таблиця 5.3 – Команди для забезпечення доступності та управління ресурсами

Команда	Призначення	Приклад використання
ulimit -t seconds	Обмеження часу виконання процесу в CPU-секундах	ulimit -t 10
ulimit -v kbytes	Обмеження віртуальної пам’яті процесу	ulimit -v 1048576
ulimit -u number	Обмеження кількості процесів користувача	ulimit -u 100
ulimit -a	Відображення всіх поточних обмежень	ulimit -a
nice -n priority command	Запуск процесу з вказаним пріоритетом (-20 до 19)	nice -n 15 ./script.sh
renice priority -p PID	Зміна пріоритету запущеного процесу	renice 10 -p 1234
systemctl set-property user.slice CPUQuota=50%	Обмеження CPU для користувацьких процесів (systemd)	systemctl set-property user-1000.slice CPUQuota=50%
iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute -j ACCEPT	Обмеження швидкості вхідних з’єднань	iptables -A INPUT -p tcp --syn --dport 80 -m connlimit --connlimit-above 20 -j REJECT

Для більш детального контролю над ресурсами у сучасних Linux-системах використовується механізм cgroups (control groups), який дозволяє ієрархічно організовувати процеси та встановлювати обмеження на рівні груп. Системи ініціалізації, такі як systemd, активно використовують cgroups для ізоляції та управління ресурсами системних сервісів та користувацьких сесій. Через systemd адміністратори можуть встановлювати політики обмеження CPU, пам’яті, дискового вводу-виведення та інших ресурсів для окремих сервісів або груп користувачів, використовуючи директиви у unit-файлах (CPUQuota, MemoryLimit, TasksMax) або динамічно через команду systemctl set-property. Такий підхід забезпечує більш гнучке та надійне управління ресурсами порівняно з традиційними обмеженнями ulimit, особливо у багатокористувацьких середовищах або при роботі з контейнеризованими застосунками.

На мережевому рівні захист від DoS-атак реалізується через комбінацію технологій, включаючи фаєрволи з підтримкою rate limiting (обмеження

швидкості запитів), системи виявлення та запобігання вторгненням (IDS/IPS), балансувальники навантаження та спеціалізовані сервіси захисту від DDoS. Фаєрвол iptables у Linux підтримує модулі для обмеження кількості з'єднань з одного IP-адреси (connlimit), обмеження швидкості пакетів (limit, hashlimit) та детектування SYN-flood атак. Правильна конфігурація таких правил дозволяє відфільтровувати аномальний трафік на ранніх стадіях, до того як він досягне прикладного рівня, суттєво знижуючи навантаження на веб-сервери та бази даних. Однак важливо знайти баланс між захистом та зручністю використання, оскільки надто жорсткі обмеження можуть заблокувати легітимних користувачів, особливо тих, хто працює через загальні NAT-шлюзи або проксі-сервери.

Високодоступні архітектури (High Availability, HA) представляють системний підхід до забезпечення доступності критичних сервісів через надмірність компонентів, автоматичне переключення на резервні системи (failover) та балансування навантаження між кількома вузлами. Типова HA-конфігурація включає кілька серверів у кластері, які можуть автоматично перебирати на себе навантаження у разі відмови одного з вузлів, з використанням розподіленої системи зберігання даних або механізмів реплікації для забезпечення узгодженості стану. Такі рішення часто базуються на спеціалізованому програмному забезпеченні, як-от Pacemaker, Corosync або Keepalived для Linux-кластерів, які моніторять стан вузлів та автоматично виконують міграцію ресурсів при детектуванні збоїв. Проектування високодоступних систем вимагає уваги до концепції single point of failure (єдиної точки відмови) та систематичного аналізу всіх компонентів інфраструктури для виявлення потенційних вузьких місць.

1.5 Взаємозв'язок властивостей CIA-триади та комплексний підхід до безпеки

Попри те що конфіденційність, цілісність та доступність розглядаються як окремі властивості безпеки, у реальних інформаційних системах вони тісно взаємопов'язані та часто впливають одна на одну як позитивно, так і негативно. Порушення однієї з властивостей може створити каскадний ефект, що призведе до компрометації інших аспектів безпеки. Наприклад, успішна атака на цілісність системних файлів може дозволити зловмиснику впровадити бекдор, який надалі використовується для порушення конфіденційності шляхом викрадення даних або для порушення доступності через запуск шкідливого коду. Аналогічно, DoS-атака, що спрямована на порушення доступності, може бути відволікаючим маневром для приховування спроб несанкціонованого доступу до конфіденційних даних або модифікації критичних файлів під час хаосу, спричиненого перевантаженням системи.

Захисні механізми також демонструють взаємозв'язок між властивостями CIA-триади, оскільки один і той самий технічний засіб може одночасно впливати на кілька аспектів безпеки. Криптографічні протоколи, такі як TLS або SSH, первинно розроблені для забезпечення конфіденційності шляхом

шифрування даних, але вони також гарантують цілісність завдяки використанню кодів автентифікації повідомлень (MAC) та автентифікацію сторін через сертифікати або криптографічні ключі. Водночас надмірне використання ресурсомістких криптографічних операцій може негативно вплинути на доступність системи, особливо на пристроях з обмеженими обчислювальними можливостями, створюючи компроміс між безпекою та продуктивністю. Системи резервного копіювання та відновлення даних забезпечують захист як цілісності (можливість відкату до незміненої версії), так і доступності (швидке відновлення після збою), але при неправильному налаштуванні можуть створювати ризики для конфіденційності, якщо резервні копії не шифруються належним чином.

Ефективна стратегія інформаційної безпеки вимагає збалансованого підходу до всіх трьох властивостей CIA-триади з урахуванням специфіки конкретної системи та її бізнес-вимог. Для різних типів систем пріоритети можуть суттєво відрізнятися: у фінансових системах критично важливою є цілісність транзакцій, у медичних інформаційних системах первинний фокус на конфіденційності пацієнтських даних, тоді як для систем реального часу або критичної інфраструктури найвищий пріоритет має доступність. Процес оцінки ризиків та проектування архітектури безпеки повинен включати систематичний аналіз загроз для кожної з властивостей, визначення потенційних векторів атак, оцінку можливих наслідків компрометації та вибір відповідних контрзаходів, які забезпечують оптимальний баланс між рівнем захисту, витратами на реалізацію та зручністю використання системи.

2 КЛЮЧОВІ ПИТАННЯ

1. Що таке CIA-триада та які три фундаментальні властивості інформаційної безпеки вона визначає? Наведіть визначення кожної з цих властивостей та поясніть їх практичне значення для захисту інформаційних систем.
2. У чому полягає різниця між симетричним та асиметричним шифруванням? Наведіть приклади алгоритмів кожного типу та поясніть, у яких сценаріях доцільно використовувати кожен з них.
3. Чому передача даних через незахищені протоколи (HTTP, FTP, Telnet) становить загрозу для конфіденційності? Які криптографічні протоколи використовуються для захисту даних “in transit” та як вони працюють?
4. Що таке криптографічна хеш-функція та які основні властивості вона повинна мати? Поясніть, чому SHA-256 вважається безпечним алгоритмом, тоді як MD5 і SHA-1 більше не рекомендуються для критичних застосувань.
5. Як працює механізм контролю цілісності на основі хешування? Опишіть послідовність дій для створення еталонного хешу файлу та подальшої перевірки його цілісності в Linux-системах.
6. Що таке DoS-атака (Denial of Service) і в чому полягає відмінність між DoS та DDoS? Наведіть приклади різних типів DoS-атак та поясніть їх вплив на доступність системи.

7. Які механізми обмеження ресурсів надає операційна система Linux для захисту від локальних DoS-атак? Поясніть різницю між м’якими (soft) та жорсткими (hard) обмеженнями ресурсів.

8. Що означає параметр CPU time у контексті команди ulimit -t та як він використовується для захисту системи від процесів, що споживають надмірні ресурси? Який сигнал отримує процес при досягненні цього ліміту?

9. Як інструмент tcpdump може бути використаний для демонстрації різниці між незахищеною (HTTP) та захищеною (SSH/SCP) передачею даних? Що саме можна побачити у перехопленому трафіку в кожному з випадків?

10. Поясніть взаємозв’язок між властивостями CIA-триади. Як порушення однієї властивості може призвести до компрометації інших? Наведіть конкретний приклад каскадного ефекту порушення безпеки.

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1. Моделювання порушення конфіденційності та захист шляхом шифрування

У сучасних інформаційних системах конфіденційність є однією з ключових властивостей безпеки, яка гарантує, що чутлива інформація доступна лише авторизованим особам. Порушення цієї властивості найчастіше відбувається через перехоплення незахищеного мережевого трафіку, коли дані передаються у відкритому вигляді. Для демонстрації такого сценарію створюється тестовий файл із чутливими даними, який спочатку передається через незахищений HTTP-сервер, а потім — через захищений протокол SCP. Паралельно виконується аналіз трафіку за допомогою мережевого аналізатора, що дозволяє наочно побачити різницю між відкритою та зашифрованою передачею. Цей етап підкреслює важливість використання криптографічних протоколів для захисту даних «in transit».

3.1.1. Створення тестового файлу з чутливими даними. Для моделювання реального сценарію використовується файл, який містить типову чутливу інформацію — наприклад, API-ключ або облікові дані. Такий підхід забезпечує наочність експерименту, оскільки подібні дані часто зберігаються у конфігураційних файлах програмного забезпечення. Файл створюється у домашній директорії користувача, що імітує звичайну практику роботи з локальними ресурсами.

```
echo "Confidential API_KEY=abc123xyz" > ~/secret.txt
```

Після виконання команди слід переконатися, що файл створено та містить очікуваний вміст. Для цього можна скористатися командою cat ~/secret.txt. Це також підтверджує, що подальші етапи будуть працювати з реальною інформацією, а не з порожнім або помилковим вмістом.

3.1.2. Налаштування та запуск незахищеного HTTP-сервера. Щоб продемонструвати вразливість незахищеної передачі, на одній з віртуальних машин (наприклад, ВМ-2) запускається простий HTTP-сервер на основі вбудованого модуля Python. Цей сервер надає доступ до файлової системи у вказаній директорії через стандартний веб-інтерфейс, що дозволяє будь-якому клієнту отримати файли без автентифікації чи шифрування. Такий підхід імітує поширений сценарій, коли розробники використовують HTTP для швидкої передачі даних, не усвідомлюючи ризиків.

```
# На ВМ-2 (приймач):
python3 -m http.server 8080
```

Сервер буде слухати на порту 8080 і надавати доступ до поточної директорії. Переконайтеся, що порт 8080 відкритий у фаєрволі (за замовчуванням у Debian він відкритий), і що IP-адреса ВМ-2 відома.

3.1.3. Передача файлу через HTTP та перехоплення трафіку. З іншої віртуальної машини (ВМ-1) виконується запит до HTTP-сервера для отримання файлу secret.txt. Одночасно на ВМ-1 запускається мережевий аналізатор tcpdump, який фіксує весь трафік на порту 8080. Оскільки HTTP не використовує шифрування, весь вміст передається у відкритому вигляді, що дозволяє легко перехопити чутливу інформацію.

```
# На ВМ-1 (клієнт):
wget http://<IP_BM2>:8080/secret.txt

# Паралельно на ВМ-1 (перехоплення):
sudo tcpdump -i any -A port 8080
```

У виводі tcpdump має бути видно рядок Confidential API_KEY=abc123xyz, що прямо свідчить про порушення конфіденційності. Це підтверджує, що незахищена передача даних у мережі є критичною вразливістю.

3.1.4. Безпечна передача файлу через SCP. Для порівняння той самий файл передається з ВМ-1 на ВМ-2 за допомогою протоколу SCP, який використовує шифрування SSH. SCP (Secure Copy) базується на протоколі SSH і забезпечує конфіденційність, цілісність та автентифікацію під час передачі. У цьому випадку мережевий аналізатор не зможе прочитати вміст файлу, оскільки весь трафік шифрується.

```
# На ВМ-1:
scp ~/secret.txt user@<IP_BM2>:/tmp/

# Паралельно на ВМ-1:
sudo tcpdump -i any -A port 22
```

У виводі tcpdump буде видно лише зашифровані пакети, які неможливо інтерпретувати без приватного ключа. Це демонструє ефективність криптографічного захисту при передачі даних.

3.2 Моделювання порушення цілісності та контроль за допомогою хешування

Цілісність — це властивість, яка гарантує, що дані не були змінені несанкціоновано під час зберігання або передачі. У реальних умовах зловмисник може модифікувати конфігураційні файли, скрипти або програмні компоненти, що призводить до неконтрольованої поведінки системи. Для виявлення таких змін використовуються криптографічні хеш-функції, які генерують унікальний «відбиток» файлу. Навіть незначна зміна вмісту призводить до кардинальної зміни хешу, що дозволяє легко виявити факт втручання. У цьому етапі студент створює файл, обчислює його SHA-256 хеш, потім вносить зміни і повторно обчислює хеш, щоб переконатися в чутливості механізму до будь-яких модифікацій.

3.2.1. Обчислення початкового хешу файлу. Для контролю цілісності спочатку обчислюється криптографічний хеш оригінального файлу за допомогою алгоритму SHA-256. Цей хеш зберігається окремо і використовується як еталон для подальшого порівняння. SHA-256 є одним із найбільш поширених і надійних алгоритмів хешування, який широко використовується в системах контролю цілісності, таких як AIDE або Tripwire.

```
sha256sum ~/secret.txt > ~/original.sha256
```

Результатом є файл original.sha256, який містить хеш у форматі <хеш> <ім’я_файлу>. Цей файл слід зберігати в безпечному місці, оскільки його компрометація робить механізм контролю цілісності марним.

3.2.2. Імітація несанкціонованої зміни файлу. Для моделювання атаки до файлу додається довільний рядок, який імітує втручання зловмисника. Наприклад, це може бути підміна API-ключа або додавання шкідливого коду. Така зміна може мати серйозні наслідки, якщо файл використовується системою без перевірки цілісності.

```
echo "Modified by attacker" >> ~/secret.txt
```

Після цього вміст файлу вже не відповідає оригіналу, що має бути виявлено на наступному етапі.

3.2.3. Перевірка цілісності за допомогою повторного хешування. Обчислюється новий хеш файлу після зміни, і виконується порівняння з оригінальним. Команда sha256sum -c автоматично перевіряє, чи збігається хеш

у файлі з поточним станом файлу. Якщо файл було змінено, команда поверне помилку.

```
sha256sum -c ~/original.sha256
```

Очікуваний результат: ~/secret.txt: FAILED. Це підтверджує, що механізм хешування успішно виявив зміну. У реальних системах такі перевірки автоматизуються і виконуються регулярно для критичних файлів.

3.3. Моделювання порушення доступності та захист через обмеження ресурсів

Доступність — це властивість, яка забезпечує готовність системи надавати ресурси авторизованим користувачам. Одним із найпоширеніших способів її порушення є DoS-атака (Denial of Service), коли зловмисник навмисно вичерпує обмежені ресурси системи (процесор, пам’ять, мережу), роблячи її недоступною для легітимних користувачів. У цьому етапі моделюється проста CPU-DoS-атака за допомогою команди yes, яка генерує нескінченний потік символів, споживаючи 100% одного ядра процесора. Для запобігання таким атакам Linux надає механізм обмеження ресурсів через ulimit, який дозволяє встановити максимальний час виконання процесу в CPU-секундах. Після досягнення ліміту процес автоматично завершується, що демонструє ефективний спосіб захисту від локальних DoS-атак.

3.3.1. Встановлення обмеження часу виконання процесу. Перед запуском потенційно шкідливого процесу встановлюється ліміт: процес матиме право виконуватися лише 5 секунд CPU time. Це означає, що навіть якщо процес буде працювати годину, він буде завершений, як тільки витратить 5 секунд процесорного часу. Такий підхід дозволяє системі залишатися стабільною, навіть у разі запуску шкідливого коду.

```
ulimit -t 5
```

Ця команда діє лише для поточної оболонки та її дочірніх процесів, тому її слід виконувати безпосередньо перед запуском тестового процесу.

3.3.2. Запуск процесу, що споживає CPU. Команда yes генерує нескінченний потік символів «y», що призводить до 100% завантаження одного ядра процесора. У реальних умовах подібну поведінку може демонструвати шкідливе програмне забезпечення або помилково написаний скрипт.

```
yes > /dev/null
```

Процес має автоматично завершитися через 5 секунд CPU time з повідомленням про сигнал SIGXCPU. Якщо процес не завершується, це може

свідчити про неправильне налаштування ulimit або те, що система не враховує CPU time коректно (наприклад, через використання іншої оболонки).

3.3.3. Перевірка результату завершення процесу. Після завершення процесу слід переконатися, що система залишилася стабільною: завантаження CPU повернулося до норми, інші процеси працюють коректно. Для цього можна скористатися командами top або htop. Це підтверджує, що механізм обмеження ресурсів ефективно захищає систему від локальних DoS-атак.

3.4. Аналіз результатів. На завершальному етапі студент узагальнює отримані результати, аналізуючи, як кожна з властивостей CIA-триади реалізується на практиці та як її порушення може вплинути на безпеку системи. Особлива увага приділяється тому, що порушення однієї властивості часто веде до компрометації інших: наприклад, втрата цілісності (підміна скрипта) може призвести до втрати конфіденційності (викрадення даних) або доступності (запуск шкідливого коду). Такий аналіз дозволяє зрозуміти, що CIA-триада — це не окремі, а взаємопов’язані компоненти комплексної моделі безпеки, і ефективний захист вимагає одночасного врахування всіх трьох аспектів.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані

результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 6

Тема: Налаштування мережевого фаєрволу iptables

Мета роботи: Опанувати принципи роботи мережевих фаєрволів в Linux, навчитися налаштовувати правила фільтрації, NAT та логування трафіку.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Історична довідка про розвиток фаєрволів в ОС Linux

Перші механізми захисту мережевого трафіку в Linux з'явилися на початку 1990-х років, коли інтернет почав стрімко розвиватися, а разом із ним — і загрози безпеці. Спочатку адміністратори використовували прості правила маршрутизації та фільтрації пакетів, засновані на утилітах на кшталт ipfwadm, яка була портована з BSD-систем. Вона дозволяла блокувати або дозволяти трафік на основі IP-адрес і портів, але мала обмежену функціональність і не підтримувала складні сценарії трансляції адрес (NAT).

На зміну ipfwadm у 1998 році прийшла більш гнучка система ipchains, яка вже підтримувала ланцюги правил (chains) і могла обробляти трафік для різних інтерфейсів. Це дозволило адміністраторам краще керувати мережевим захистом, особливо в корпоративних середовищах, де була потреба в розділенні трафіку між внутрішньою та зовнішньою мережами. Однак ipchains все ще не мала повноцінної підтримки NAT, що ускладнювало її використання для маскування мереж (MASQUERADE).

Справжнім проривом став вихід iptables у 2001 році, який інтегрувався в ядро Linux 2.4 і став стандартним інструментом фільтрації трафіку на багато років. Він запровадив модульну архітектуру, підтримку станів з'єднань (stateful firewall), повноцінний NAT та розширені можливості логування. Завдяки цьому iptables міг не просто блокувати пакети, а й аналізувати їх у контексті встановлених з'єднань, що значно підвищило рівень безпеки.

У 2010-х роках з'явилися нові технології, такі як nftables, яка повинна була замінити iptables через більш простий синтаксис і кращу продуктивність. Однак через звичку адміністраторів і велику кількість існуючих конфігурацій iptables залишався основним інструментом до середини 2020-х. Паралельно розвивалися і фаєрволи нового покоління, такі як firewalld, який використовує зони (zones) для спрощення управління правилами, і bpfilter, що працює на рівні ядра для пришвидшення обробки трафіку.

Сьогодні Linux пропонує цілий спектр рішень для мережевої безпеки — від класичних iptables до сучасних nftables та eBPF-фаєрволів. Кожен із них має свої переваги: iptables досі актуальний через свою універсальність, nftables пропонує кращу продуктивність, а firewalld спрощує управління для початківців. Розвиток цих технологій показує, як Linux еволюціонував від простих фільтрів пакетів до потужних систем захисту, здатних протистояти сучасним кіберзагрозам.

Таблиця 6.1 – Порівняльна таблиця фаєрволів в Linux

Фаєрвол	Рік появи	Основні можливості	Недоліки	Сучасний статус
ipfwadm	1994	Базова фільтрація за IP/портами	Немає NAT, обмежений функціонал	Застарів
ipchains	1998	Ланцюги правил, покращена фільтрація	Слабка підтримка NAT	Замінено на iptables
iptables	2001	Повноцінний NAT, stateful інспекція, модульність	Складний синтаксис	Досі використовується
nftables	2014	Простий синтаксис, висока продуктивність	Повільний перехід з iptables	Активно розвивається
firewalld	2011	Зони, динамічне управління	Менша гнучкість для складних правил	Популярний у дистрибутивах (RHEL/Fedora)
bpfilter/eBPF	2018	Робота на рівні ядра, висока швидкість	Складність налаштування	Перспективна технологія

1.2 Структура iptables та процес обробки мережевих пакетів

Схема, наведена на рисунку, ілюструє архітектуру фаєрволу iptables в Linux, яка базується на трьох основних таблицях: *filter* (фільтрація), *nat* (трансляція мережевих адрес) та *mangle* (модифікація заголовків пакетів). Кожна таблиця містить набір ланцюгів (chains), через які проходять пакети на різних етапах обробки. Вхідний трафік спочатку потрапляє на мережевий інтерфейс (наприклад, eth0), після чого проходить через ланцюг PREROUTING, де відбувається модифікація заголовків (mangle) або трансляція адрес (nat). Далі система визначає, чи призначений пакет для локального процесу, чи має бути перенаправлений далі (forwarding), і відповідно направляє його через ланцюги INPUT, FORWARD або OUTPUT. Після обробки в ланцюгах POSTROUTING пакет відправляється через вихідний інтерфейс у мережу.

Пакет від вхідного інтерфейсу до локального додатку. Коли пакет надходить з зовнішньої мережі на вхідний інтерфейс (наприклад, eth0), він спочатку проходить через ланцюг PREROUTING у таблиці mangle, де можна змінити параметри, такі як TTL або DSCP. Потім у таблиці nat у цьому ж ланцюгу може відбутися DNAT (Destination NAT), якщо потрібно перенаправити пакет на іншу адресу всередині локальної мережі. Після цього система перевіряє таблицю маршрутизації, щоб визначити, що пакет призначений для локального процесу, і направляє його через ланцюг INPUT у таблиці filter, де застосовуються правила блокування або дозволу трафіку. Якщо пакет успішно проходить усі перевірки, він передається локальному додатку, який обробляє відповідний порт (наприклад, веб-сервер на порту 80).

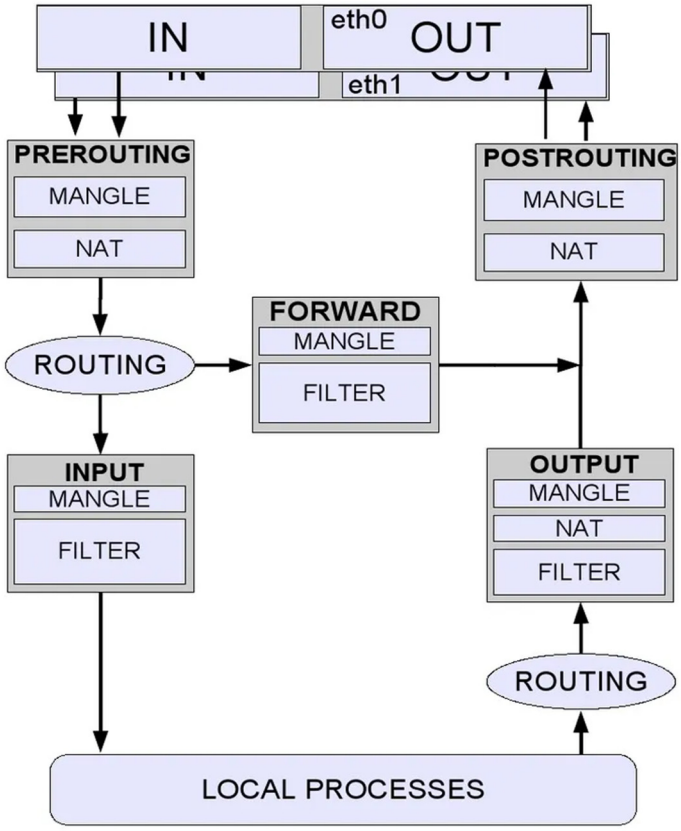

Рисунок 6.1 – Архітектура фаєрволу iptables

Транзитний пакет (forwarding). Транзитний трафік, який не призначений для локальної системи, але має бути перенаправлений через неї (наприклад, у випадку маршрутизатора або VPN-сервера), проходить аналогічний початковий шлях через PREROUTING у таблицях mangle та nat. Однак після маршрутизації система визначає, що пакет має бути перенаправлений, і направляє його через ланцюг FORWARD у таблиці filter, де застосовуються правила, що регулюють передачу між інтерфейсами. На цьому етапі можна блокувати небажаний трафік або обмежувати швидкість передачі. Після успішної фільтрації пакет проходить через ланцюг POSTROUTING, де в таблиці nat може бути застосований SNAT (Source NAT) для маскування внутрішньої адреси, перш ніж він буде відправлений у зовнішню мережу через вихідний інтерфейс.

Пакет від локального додатку до вихідного інтерфейсу. Коли локальний додаток (наприклад, браузер або SSH-клієнт) генерує вихідний трафік, пакет спочатку проходить через ланцюг OUTPUT у таблиці mangle, де можна змінити його параметри. Потім у таблиці filter застосовуються правила, які визначають, чи дозволено відправляти цей трафік. Якщо пакет успішно проходить фільтрацію, він направляється до ланцюга POSTROUTING у таблиці nat, де може бути застосований SNAT (наприклад, MASQUERADE для динамічної зміни IP-адреси вихідного інтерфейсу). Після цього пакет передається через вихідний інтерфейс (наприклад, eth1 або ppp0) у зовнішню мережу. Цей процес забезпечує безпеку та контроль над вихідним трафіком, запобігаючи несанкціонованим з’єднанням.

Схема демонструє, як iptables системно обробляє різні типи трафіку, забезпечуючи гнучкість у налаштуванні правил безпеки. Кожен етап обробки пакету — від вхідного інтерфейсу до локальних процесів, транзитного перенаправлення чи вихідного трафіку — має свої ланцюги та таблиці, що дозволяє точно керувати мережевим захистом. Розуміння цієї структури є ключовим для адміністраторів, які налаштовують фаєрволи в Linux.

1.3 Ключі командного рядка iptables

Iptables — це потужний інструмент для налаштування фаєрволу в Linux, який дозволяє керувати мережевим трафіком за допомогою правил, що застосовуються до пакетів. Основна робота з iptables відбувається через командний рядок, де використовуються спеціальні ключі для створення, редагування, перевірки та видалення правил. Нижче наведено детальний опис основних ключів iptables разом із прикладами їх використання.

Таблиця 6.2 – Основні ключі для управління правилами

Ключ	Пояснення	Приклад
-P (--policy)	Встановлює політику за замовчуванням для ланцюжка (ACCEPT, DROP, REJECT).	iptables -P INPUT DROP
-A (--append)	Додає правило в кінець ланцюжка.	iptables -A INPUT -s 192.168.1.1 -j DROP
-I (--insert)	Вставляє правило на початок ланцюжка (або за номером).	iptables -I INPUT 2 -p tcp --dport 22 -j ACCEPT
-D (--delete)	Видаляє правило за номером або вмістом.	iptables -D INPUT 3
-R (--replace)	Замінює правило за номером.	iptables -R INPUT 1 -s 10.0.0.1 -j ACCEPT
-L (--list)	Виводить список правил.	iptables -L -v
-F (--flush)	Очищає всі правила в ланцюжку (або всі ланцюжки).	iptables -F INPUT
-N (--new-chain)	Створює новий користувацький ланцюжок.	iptables -N MY_CHAIN
-X (--delete-chain)	Видаляє порожній користувацький ланцюжок.	iptables -X MY_CHAIN
-Z (--zero)	Обнуляє лічильники пакетів і байтів.	iptables -Z

Таблиця 6.3 – Ключі для фільтрації трафіку

Ключ	Пояснення	Приклад
-p (--protocol)	Вказує протокол (tcp, udp, icmp, all).	iptables -A INPUT -p tcp --dport 80 -j ACCEPT
-s (--source)	Фільтрує за IP-адресою джерела.	iptables -A INPUT -s 192.168.1.100 -j DROP
-d (--destination)	Фільтрує за IP-адресою призначення.	iptables -A OUTPUT -d 8.8.8.8 -j REJECT
-i (--in-interface)	Вказує вхідний інтерфейс.	iptables -A INPUT -i eth0 -j ACCEPT
-o (--out-interface)	Вказує вихідний інтерфейс.	iptables -A OUTPUT -o eth1 -j DROP

Ключ	Пояснення	Приклад
--sport (--source-port)	Фільтрує за портом джерела.	iptables -A INPUT -p tcp --sport 53 -j ACCEPT
--dport (--destination-port)	Фільтрує за портом призначення.	iptables -A INPUT -p tcp --dport 22 -j ACCEPT
-m (--match)	Використовує додаткові модулі (state, multiport, limit тощо).	iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT

Таблиця 6.4 – Ключі для дій з пакетами (-j)

Ключ	Пояснення	Приклад
-j ACCEPT	Дозволити пакет.	iptables -A INPUT -j ACCEPT
-j DROP	Відкинути пакет без повідомлення.	iptables -A INPUT -s 10.0.0.1 -j DROP
-j REJECT	Відхилити пакет із повідомленням (ICMP unreachable).	iptables -A INPUT -p tcp --dport 23 -j REJECT
-j LOG	Логувати пакет у системний журнал (/var/log/syslog).	iptables -A INPUT -j LOG --log-prefix "Blocked: "
-j DNAT	Перенаправляє пакет (Destination NAT).	iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to 192.168.1.10:8080
-j SNAT	Змінює IP-джерело (Source NAT).	iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4
-j MASQUERADE	Автоматичний NAT для динамічних IP (наприклад, PPPoE).	iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE

Таблиця 6.5 – Додаткові корисні ключі

Ключ	Пояснення	Приклад
-t (--table)	Вказує таблицю (filter, nat, mangle, raw).	iptables -t nat -L
-v (--verbose)	Детальний вивід (разом з лічильниками).	iptables -L -v
-n (--numeric)	Виводить IP та порти у числовому форматі.	iptables -L -n
--line-numbers	Показує номери правил (для видалення/заміни).	iptables -L --line-numbers

Приклад комплексного налаштування:

```
# Дозволити локальний трафік
iptables –A INPUT –i lo –j ACCEPT
iptables –A OUTPUT –o lo –j ACCEPT

# Дозволити вхідні SSH та HTTP
iptables –A INPUT –p tcp --dport 22 –j ACCEPT
iptables –A INPUT –p tcp --dport 80 –j ACCEPT

# Блокувати всі інші вхідні з’єднання
iptables –P INPUT DROP

# Дозволити вихідний трафік
iptables –P OUTPUT ACCEPT
```

```
# Налаштувати NAT для виходу в інтернет
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Ці приклади охоплюють основні сценарії використання iptables. Для глибшого вивчення варто ознайомитися з офіційною документацією (man iptables).

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Встановлення пакета iptables-persistent

У Linux правила фаєрволу, налаштовані за допомогою iptables, зберігаються лише в оперативній пам’яті ядра і втрачаються після перезавантаження системи. Щоб уникнути необхідності повторного налаштування правил після кожного перезапуску, використовується пакет iptables-persistent (у новіших версіях Debian — netfilter-persistent). Цей пакет автоматично зберігає поточну конфігурацію iptables у файли /etc/iptables/rules.v4 (для IPv4) та /etc/iptables/rules.v6 (для IPv6) і відновлює їх під час завантаження системи. На цьому етапі студент встановлює цей пакет, щоб забезпечити стійкість налаштованих правил безпеки.

```
sudo apt update
sudo apt install iptables-persistent
```

Під час встановлення система запропонує зберегти поточні правила IPv4 та IPv6. На цьому етапі можна відповісти «Ні», оскільки правила ще не налаштовані. Після встановлення слід переконатися, що служба netfilter-persistent активна:

```
sudo systemctl status netfilter-persistent
```

Якщо служба працює, це свідчить про те, що система готова зберігати та відновлювати правила фаєрволу. Файли /etc/iptables/rules.v4 та /etc/iptables/rules.v6 будуть створені автоматично після першого збереження конфігурації.

3.2 Налаштування правил iptables згідно з індивідуальним варіантом

На цьому етапі студент реалізує конкретний сценарій безпеки, наведений у таблиці варіантів лабораторної роботи. Кожен варіант містить три типи завдань: - блокування або дозвіл вхідного трафіку (INPUT), - керування вихідним трафіком (OUTPUT), - фільтрація транзитного трафіку (FORWARD).

Таблиця 6.6 – Варіанти індивідуальних завдань

Варіант	Завдання
1	Заборонити вхідні з'єднання на порт 23; Дозволити вихідні з'єднання на порт 80; Відкинути транзитні пакети з IP-адреси 203.0.113.1
2	Дозволити вхідні з'єднання з підмережі 10.0.0.0/8; Заборонити вихідні з'єднання до порту 21; Відкинути транзитні пакети з встановленим флагом PSH
3	Заборонити вхідні пакети з інтерфейсу wlan0; Дозволити вихідні з'єднання на порт 587; Відкинути транзитні пакети з неправильною контрольної сумою
4	Дозволити вхідні з'єднання до 192.168.1.1 на порт 8080; Заборонити вихідні з'єднання протоколу L2TP; Відкинути транзитні пакети з пакунками типу ICMP redirect
5	Заборонити вхідні з'єднання з IP 192.0.2.55; Дозволити вихідні з'єднання на порти 20-21; Відкинути транзитний трафік з VLAN ID 200
6	Дозволити вхідні з'єднання протоколу TCP на порт 3306; Заборонити вихідні з'єднання до мережі 172.20.0.0/14; Відкинути транзитні пакети з TOS 0x08
7	Заборонити вхідні з'єднання на порт 25; Дозволити вихідні з'єднання до 8.8.8.8; Відкинути транзитні пакети з пакунками типу Source Route
8	Дозволити вхідні з'єднання на порт 53 (UDP); Заборонити вихідні з'єднання до IP 198.51.100.100; Відкинути транзитні пакети з TCP-флагом ACK без встановленого SYN
9	Заборонити вхідні з'єднання на порт 135; Дозволити вихідні HTTPS-запити; Відкинути транзитний трафік з MAC-адреси 00:DE:AD:BE:EF:00
10	Дозволити вхідні з'єднання з IP 10.10.10.10 на порт 22; Заборонити вихідні з'єднання протоколу ESP; Відкинути транзитні пакети розміром менше 100 байт
11	Заборонити вхідні з'єднання на порт 21; Дозволити вихідні з'єднання на порт 443; Відкинути транзитний трафік з мережі 10.10.0.0/16
12	Дозволити вхідні з'єднання з IP 172.16.0.15; Заборонити вихідні з'єднання до порту 22; Відкинути всі вхідні UDP-пакети
13	Заборонити вхідний трафік на інтерфейсі eth2; Дозволити вихідні ICMP-запити; Відкинути транзитні пакети до порту 5900
14	Дозволити вхідні з'єднання на порт 3389 лише з 192.168.0.10; Заборонити вихідні з'єднання до 45.33.12.10; Відкинути транзитний трафік протоколу SCTP
15	Заборонити вхідні пакети з MAC-адреси 00:1A:2B:3C:4D:5E; Дозволити вихідні з'єднання на порт 123; Відкинути транзитні пакети зі статусом NEW
16	Дозволити вхідні з'єднання лише для встановлених з'єднань; Заборонити вихідні з'єднання на порти 27000-27050; Відкинути транзитний трафік між eth0 та eth1
17	Заборонити вхідні з'єднання з підмережі 192.168.100.0/24; Дозволити вихідні DNS-запити; Відкинути транзитні пакети з TTL менше 5
18	Дозволити вхідні з'єднання на порт 8443; Заборонити вихідні з'єднання протоколу IGMP; Відкинути транзитні пакети з фрагментацією
19	Заборонити вхідні пакети з портом джерела 53; Дозволити вихідні з'єднання до 1.0.0.1; Відкинути транзитний трафік вночі (22:00-06:00)
20	Дозволити вхідні з'єднання на порт 9090; Заборонити вихідні з'єднання до 185.143.223.0/24; Відкинути транзитні пакети з встановленим флагом SYN
21	Заборонити вхідні пакети з типом служби Minimize-Delay; Дозволити вихідні з'єднання на порт 995; Відкинути транзитні пакети розміром більше 1500 байт
22	Дозволити вхідні з'єднання з VLAN ID 100; Заборонити вихідні з'єднання до порту 5060; Відкинути транзитні пакети з IP option
23	Заборонити вхідні пакети з портом призначення 31337; Дозволити вихідні з'єднання до 9.9.9.9; Відкинути транзитні пакети з нульовим TTL
24	Дозволити вхідні з'єднання на порт 2222; Заборонити вихідні з'єднання протоколу

	GRE; Відкинути транзитні пакети з встановленим RST флагом
25	Заборонити вхідні пакети з IP 169.254.0.0/16; Дозволити вихідні з'єднання на порт 1194; Відкинути транзитні пакети з MSS більше 1400
26	Дозволити вхідні з'єднання з мітки 0x1; Заборонити вихідні з'єднання до порту 1900; Відкинути транзитні пакети з checksum помилкою
27	Заборонити вхідні пакети з TOS 0x10; Дозволити вихідні з'єднання до 208.67.222.222; Відкинути транзитні пакети з неправильним TCP sequence
28	Дозволити вхідні з'єднання на порт 4567; Заборонити вихідні з'єднання до 224.0.0.0/4; Відкинути транзитні пакети з встановленим URG флагом
29	Заборонити вхідні пакети з портом джерела 1024-65535; Дозволити вихідні з'єднання на порт 636; Відкинути транзитні пакети з неправильним IP заголовком
30	Дозволити вхідні з'єднання з DSCP AF21; Заборонити вихідні з'єднання до порту 161; Відкинути транзитні пакети з встановленим FIN флагом

Правила формулюються з урахуванням принципів роботи iptables: порядку обробки пакетів, типів ланцюгів (INPUT, OUTPUT, FORWARD), використання ключів (-s, -d, -p, --dport, -i, -o тощо) та дій (-j DROP, -j ACCEPT, -j REJECT). Особливу увагу слід приділити тому, що правила застосовуються послідовно, і перше збігове правило визначає долю пакета. Тому критичні правила (наприклад, дозвіл SSH) мають бути розміщені вище загальних правил блокування.

Наприклад, для варіанту 1 («Заборонити вхідні з’єднання на порт 23; Дозволити вихідні з’єднання на порт 80; Відкинути транзитні пакети з IP-адреси 203.0.113.1») виконуються такі команди (замість цих команд студент виконує правила, відповідні своєму варіанту з таблиці):

```
# Блокування вхідних з'єднань на порт 23 (Telnet)
sudo iptables -A INPUT -p tcp --dport 23 -j DROP

# Дозвіл вихідних HTTP-запитів
sudo iptables -A OUTPUT -p tcp --dport 80 -j ACCEPT

# Блокування транзитного трафіку з конкретної IP-адреси
sudo iptables -A FORWARD -s 203.0.113.1 -j DROP
```

Після налаштування всіх правил рекомендується переглянути їх у числовому форматі, щоб уникнути помилок інтерпретації доменних імен або служб:

```
sudo iptables -L -n --line-numbers
```

У виводі слід перевірити: - чи присутні всі три правила, - чи правильно вказані IP-адреси, порти та напрямки трафіку, - чи немає конфліктів з існуючими правилами (наприклад, загальний DROP вище дозволу).

3.3. Збереження правил фаєрволу для автозавантаження

Після налаштування правил їх необхідно зберегти у постійну конфігурацію, щоб вони залишилися після перезавантаження. Для цього

використовується команда netfilter-persistent save, яка записує поточний стан iptables у файли /etc/iptables/rules.v4 та /etc/iptables/rules.v6. Це гарантує, що при наступному завантаженні системи всі правила будуть автоматично відновлені службою netfilter-persistent.

```
sudo netfilter-persistent save
```

Після виконання команди слід переконатися, що файли конфігурації були оновлені:

```
cat /etc/iptables/rules.v4
```

У виводі має відображатися текстова версія всіх налаштованих правил у форматі, сумісному з iptables-restore. Якщо файл порожній або містить лише коментарі, це означає, що правила не були збережені — у такому випадку слід повторити команду збереження.

Для додаткової перевірки можна виконати симуляцію перезавантаження: очистити всі правила, а потім відновити їх з файлу:

```
sudo iptables -F
sudo iptables-restore < /etc/iptables/rules.v4
sudo iptables -L -n
```

Якщо правила з’явилися знову — конфігурація збережена коректно. Правила, які не збережені через iptables-persistent, зникнуть після перезапуску системи!

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної

системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 7

Тема: Шифрування файлів за допомогою GPG

Мета роботи: Опанувати процеси створення ключових пар, шифрування даних відкритим ключем, передачі зашифрованих файлів через SSH та їх подальшого розшифрування.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Основи криптографії та безпечного обміну даними

Криптографія є важливою складовою сучасних систем захисту інформації, яка забезпечує конфіденційність, цілісність та автентичність даних шляхом їх перетворення у формат, недоступний для несанкціонованого доступу. Історично криптографія розвивалася протягом тисячоліть, еволюціонувавши від простих методів заміни символів до складних математичних алгоритмів, що використовують обчислювально складні проблеми як основу для захисту інформації. Сучасна криптографія поділяється на дві основні категорії: симетричну та асиметричну, кожна з яких має свої переваги, недоліки та області застосування у світі інформаційних технологій. Симетрична криптографія використовує один і той самий ключ для шифрування та розшифрування даних, що робить її швидкою, але створює проблему безпечного розподілу ключів між учасниками комунікації. Асиметрична криптографія вирішує цю проблему шляхом використання пари ключів - відкритого та закритого, дозволяючи безпечний обмін даними без попереднього узгодження секретного ключа між сторонами.

Асиметрична криптографія, також відома як криптографія з відкритим ключем, грунтується на математичних функціях, які легко обчислюються в одному напрямку, але обчислювально складні для зворотного перетворення без наявності додаткової інформації (закритого ключа). Алгоритм RSA, розроблений Рівестом, Шаміром та Адлеманом у 1977 році, є одним з перших і найбільш поширених алгоритмів асиметричної криптографії, що базується на складності факторизації (розкладання на множники) великих чисел. Принцип роботи асиметричного шифрування полягає в тому, що повідомлення, зашифроване відкритим ключем, може бути розшифроване тільки відповідним закритим ключем, який має залишатися в таємниці. Відкритий ключ може бути вільно розповсюджений і використовується для шифрування повідомлень або перевірки цифрових підписів, тоді як закритий ключ зберігається в таємниці та використовується для розшифрування повідомлень або створення цифрових підписів. Цей механізм забезпечує не тільки конфіденційність повідомлень, але й можливість автентифікації відправника за допомогою цифрового підпису.

Таблиця 7.1 – Таблиця порівняння симетричного та асиметричного шифрування

Характеристика	Симетричне шифрування	Асиметричне шифрування
Ключі	Один спільний ключ	Пара ключів: відкритий та закритий
Швидкість	Висока	Низька (до 1000 разів повільніше)
Розмір ключа	Менший (128-256 біт)	Більший (2048-4096 біт)
Безпека	Залежить від секретності ключа	Залежить від секретності закритого ключа
Обмін ключами	Потребує захищеного каналу	Не потребує захищеного каналу
Масштабованість	Низька (n×(n-1)/2 ключів для n користувачів)	Висока (2×n ключів для n користувачів)
Автентифікація	Складна реалізація	Вбудована через цифровий підпис
Приклади алгоритмів	AES, 3DES, Blowfish	RSA, DSA, ECC

GNU Privacy Guard (GnuPG або GPG) є вільною програмною реалізацією стандарту OpenPGP, яка забезпечує шифрування та підписування даних, що робить її важливим інструментом для захисту електронної пошти, файлів та інших форм цифрової комунікації в середовищі Linux. GnuPG підтримує різні алгоритми шифрування, включаючи симетричні (AES, 3DES, Blowfish) та асиметричні (RSA, DSA, ElGamal), а також алгоритми хешування (SHA-256, SHA-384, SHA-512), що дозволяє користувачам вибирати рівень захисту відповідно до їхніх потреб. Важливою особливістю GnuPG є децентралізована модель довіри, яка дозволяє користувачам самостійно визначати рівень довіри до ключів інших користувачів, що є альтернативою централізованій інфраструктурі відкритих ключів (PKI). GnuPG має широкий спектр застосувань, включаючи шифрування файлів, електронної пошти, створення цифрових підписів, а також верифікацію цілісності програмного забезпечення та захищені комунікації між користувачами, що робить її невід'ємною частиною набору інструментів з безпеки для користувачів Linux.

Процес шифрування та обміну зашифрованими даними з використанням GnuPG включає кілька ключових етапів. Спочатку користувач генерує пару ключів (відкритий та закритий) за допомогою команди *gpg --gen-key*, під час якої вказуються персональні дані, такі як ім'я та електронна пошта, а також параметри ключа, включаючи тип алгоритму, розмір ключа та термін дії. Після створення ключів відкритий ключ може бути експортований за допомогою команди *gpg --export* та переданий іншим користувачам через електронну пошту, веб-сайт або сервер ключів, тоді як закритий ключ залишається на пристрої користувача та повинен бути захищений надійним паролем. Для шифрування файлу відправник використовує відкритий ключ отримувача за допомогою команди *gpg --encrypt --recipient user@example.com filename*, в результаті чого створюється зашифрований файл з розширенням *.gpg*, який може бути переданий отримувачу будь-яким зручним способом. При отриманні зашифрованого файлу отримувач використовує свій закритий ключ для

розшифрування за допомогою команди *gpg --decrypt filename.gpg*, після чого вміст файлу стає доступним. Цей процес забезпечує конфіденційність передачі даних, оскільки навіть якщо зашифрований файл буде перехоплений, без відповідного закритого ключа його неможливо розшифрувати.

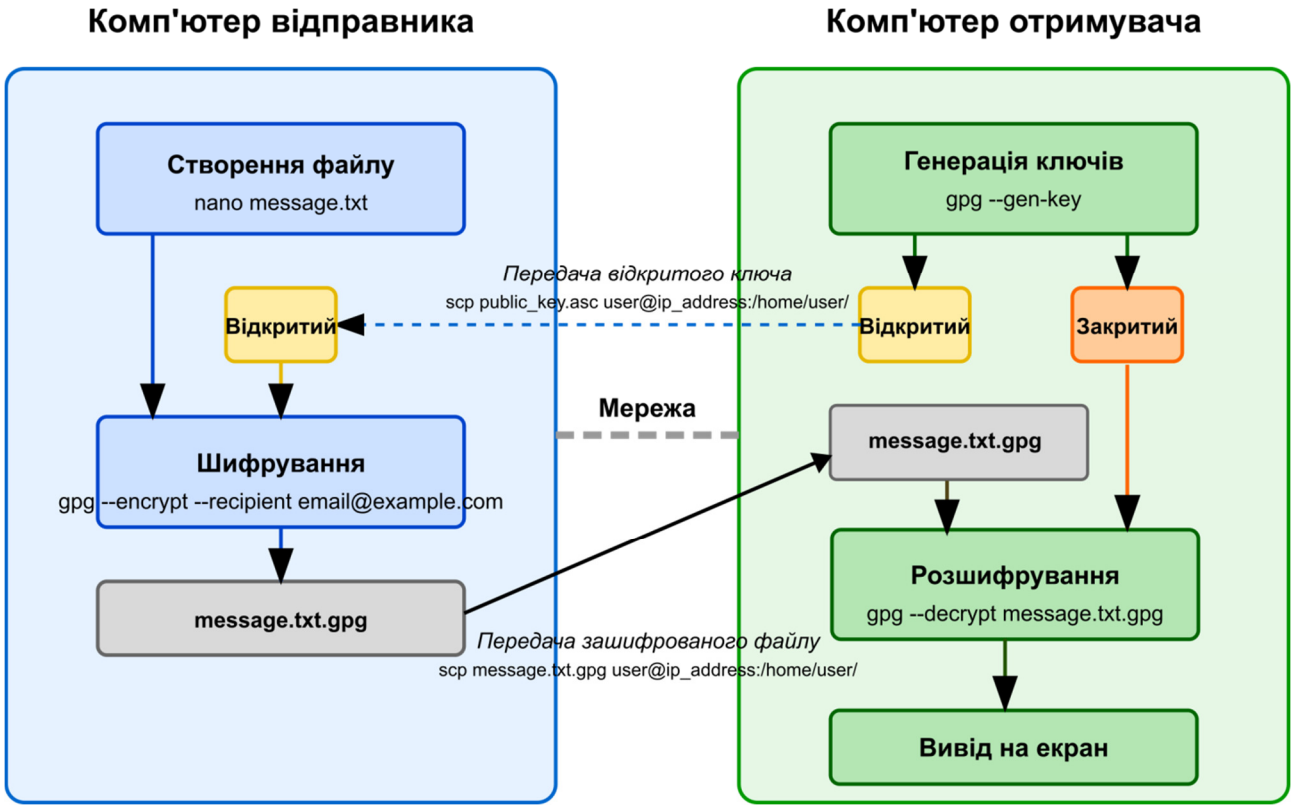

Рисунок 7.1 – Цикл безпечного обміну даними з використанням асиметричної криптографії в середовищі Linux

Наведений рисунок демонструє повний цикл безпечного обміну даними з використанням асиметричної криптографії в середовищі Linux, де весь процес розпочинається з генерації ключової пари (відкритого та закритого) на комп'ютері отримувача за допомогою команди *gpg --gen-key*, після чого відкритий ключ безпечно передається на комп'ютер відправника через SSH/SCP, що є фундаментальною передумовою для подальших операцій. Відправник, отримавши відкритий ключ, створює текстовий файл з довільним вмістом, який потім шифрується саме цим відкритим ключем отримувача за допомогою команди *gpg --encrypt --recipient email@example.com*, в результаті чого формується зашифрований файл з розширенням *.gpg*, захищений від несанкціонованого доступу. Далі зашифрований файл передається через мережу від відправника до отримувача, знову ж таки використовуючи захищений протокол передачі SSH/SCP, що забезпечує додатковий рівень захисту під час транспортування даних. Отримувач, володіючи відповідним закритим ключем (який ніколи не покидав його комп'ютер), здійснює розшифрування отриманого файлу командою *gpg --decrypt message.txt.gpg*, застосовуючи свій закритий ключ, після чого розшифрований вміст виводиться на екран і стає доступним для читання лише легітимному отримувачу. Ця схема

наочно ілюструє ключову особливість асиметричної криптографії: інформація, зашифрована відкритим ключем певного користувача, може бути розшифрована виключно відповідним закритим ключем того ж користувача, що забезпечує конфіденційність даних навіть при відкритій передачі зашифрованих файлів та відкритих ключів через публічні мережі.

Таблиця 7.2 – Таблиця основних команд програми GPG

Команда	Опис	Приклад використання
--gen-key	Генерація нової пари ключів	gpg --gen-key
--list-keys	Перегляд доступних відкритих ключів	gpg --list-keys
--list-secret-keys	Перегляд доступних закритих ключів	gpg --list-secret-keys
--export	Експорт відкритого ключа	gpg --export --armor user@example.com > public_key.asc
--export-secret-keys	Експорт закритого ключа	gpg --export-secret-keys --armor user@example.com > private_key.asc
--import	Імпорт ключа	gpg --import key_file.asc
--encrypt	Шифрування файлу	gpg --encrypt --recipient user@example.com file.txt
--decrypt	Розшифрування файлу	gpg --decrypt file.txt.gpg > decrypted_file.txt
--sign	Підписування файлу	gpg --sign file.txt
--verify	Перевірка підпису	gpg --verify file.txt.sig
--clearsign	Створення підпису у відкритому текстовому форматі	gpg --clearsign file.txt
--symmetric	Симетричне шифрування файлу	gpg --symmetric file.txt
--edit-key	Редагування ключа	gpg --edit-key user@example.com
--delete-key	Видалення відкритого ключа	gpg --delete-key user@example.com
--delete-secret-key	Видалення закритого ключа	gpg --delete-secret-key user@example.com

Безпека асиметричної криптографії залежить від кількох критичних факторів, дотримання яких є необхідним для ефективного захисту інформації. Довжина ключа є важливим параметром, що визначає стійкість криптосистеми до атак методом прямого перебору або інших методів криптоаналізу, причому сучасні рекомендації пропонують використовувати ключі RSA довжиною не менше 2048 біт для адекватного рівня захисту на найближчі роки. Захист закритого ключа є абсолютно критичним, оскільки його компрометація дозволяє зловмиснику розшифровувати всі повідомлення, зашифровані відповідним відкритим ключем, або створювати підроблені цифрові підписи від імені власника ключа. Для захисту закритого ключа рекомендується використовувати сильний пароль, зберігати ключ на окремому фізичному носії (наприклад, смарт-карті або USB-токені), регулярно створювати резервні копії та оновлювати ключі після певного періоду використання. Перевірка автентичності відкритих ключів також є важливим аспектом безпеки, оскільки використання підробленого відкритого ключа може призвести до шифрування даних для зловмисника замість справжнього отримувача, для чого

використовуються такі механізми як цифрові сертифікати, мережа довіри PGP або особиста верифікація відбитка ключа. Дотримання цих принципів безпеки дозволяє забезпечити надійний захист інформації при використанні асиметричної криптографії.

Інтеграція GnuPG з іншими інструментами безпеки Linux, такими як SSH (Secure Shell), створює потужну екосистему для захищеного віддаленого доступу та передачі файлів. SSH за замовчуванням використовує пару ключів для автентифікації, що дозволяє встановлювати захищені з'єднання між комп'ютерами без введення паролів, а команда SCP (Secure Copy), яка базується на протоколі SSH, забезпечує безпечну передачу файлів між системами. Комбінування GnuPG та SSH/SCP дозволяє створити багаторівневу систему захисту, де файли спочатку шифруються за допомогою GnuPG, а потім передаються через захищений канал SSH, що значно підвищує загальний рівень безпеки. Автоматизація процесів шифрування та передачі файлів може бути реалізована за допомогою скриптів, що ще більше спрощує роботу з зашифрованими даними. Розуміння взаємодії між цими інструментами є важливим для створення ефективної стратегії захисту інформації в середовищі Linux, особливо для організацій, які працюють з конфіденційними даними або в розподілених системах, де безпечна передача інформації є критичною вимогою.

2 КЛЮЧОВІ ПИТАННЯ

1. У чому полягає принципова різниця між симетричним і асиметричним шифруванням, і які переваги дає використання асиметричної криптографії при обміні даними через незахищені канали?
2. Як реалізується механізм конфіденційності в асиметричній криптографії: яким ключем шифрується повідомлення і яким — розшифровується? Чому це забезпечує безпеку навіть при публічному розповсюдженні відкритого ключа?
3. Які основні етапи необхідно виконати для створення, експорту та імпорту ключової пари за допомогою GnuPG, і які параметри слід вказувати під час генерації ключа для забезпечення його стійкості?
4. Чому в процесі шифрування використовується саме відкритий ключ отримувача, а не відправника? Як це запобігає несанкціонованому доступу до даних третіми особами?
5. Які команди GnuPG використовуються для шифрування та розшифрування файлів, і як забезпечується автентифікація користувача під час розшифрування (через що запитується пароль)?
6. Яким чином інтеграція GnuPG з протоколом SSH/SCP підвищує загальний рівень безпеки при передачі конфіденційних даних у мережі?
7. Які фактори визначають стійкість асиметричної криптосистеми (наприклад, RSA), і чому рекомендується використовувати ключі довжиною не менше 2048 біт?
8. Чому закритий ключ ніколи не повинен покидати пристрій власника, і які наслідки може мати його компрометація?

9. Як можна перевірити автентичність отриманого відкритого ключа, щоб уникнути атаки «людина посередині» (Man-in-the-Middle)?

10. Які додаткові можливості надає GnuPG крім шифрування (наприклад, цифровий підпис), і як вони реалізують функції цілісності та автентифікації даних?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1. Підготовка віртуального середовища для безпечного обміну даними

Перед початком практичної роботи з асиметричної криптографії необхідно створити ізольоване мережеве середовище, що складається з двох віртуальних машин (ВМ), які будуть моделювати ролі відправника та отримувача зашифрованих даних. Ця підготовча фаза включає налаштування мережевих параметрів, забезпечення статичних IP-адрес, унікальних імен хостів та перевірку зв'язності між машинами. Такий підхід імітує реальну інфраструктуру, де передача конфіденційної інформації відбувається через ненадійну мережу, і тому вимагає надійного шифрування. Успішна підготовка цього середовища є необхідною умовою для подальшого тестування механізмів шифрування та розшифрування за допомогою GnuPG.

3.1.1 Налаштування мережевих параметрів першої віртуальної машини. Для забезпечення стабільної взаємодії між віртуальними машинами необхідно налаштувати статичні IP-адреси замість динамічних, які можуть змінюватися після перезавантаження. Спочатку визначається адресний діапазон локальної мережі, до якої належить хост-система. Для цього виконується сканування підмережі за допомогою інструменту nmap, що дозволяє побачити всі активні IP-адреси. Після цього віртуальна машина налаштовується на отримання фіксованої адреси з цієї підмережі, що гарантує постійну доступність для іншої ВМ. Конфігурація виконується через файл Netplan, який є стандартним інструментом управління мережею в сучасних версіях Debian.

```
# Визначення IP-діапазону хост-системи (приклад для Windows):
# ipconfig

# Сканування підмережі (замініть 192.168.2.0/24 на вашу
підмережу):
nmap -sn 192.168.2.0/24

# Редагування конфігурації мережі:
sudo nano /etc/netplan/01-netcfg.yaml
```

У файл */etc/netplan/01-netcfg.yaml* додається наступний вміст (з урахуванням вашої підмережі):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.2.60/24]
      routes:
        - to: default
          via: 192.168.2.1
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]
```

Після збереження файлу застосовується нова конфігурація:

```
sudo netplan apply
```

Для перевірки коректності налаштувань виконується команда *ip a*, яка має показати статичну IP-адресу 192.168.2.60 на інтерфейсі *enp0s3*. Також рекомендується виконати ping 8.8.8.8, щоб переконатися в наявності доступу до Інтернету.

3.1.2 Налаштування другої віртуальної машини. Щоб уникнути конфліктів IP-адрес та забезпечити чітку ідентифікацію, друга віртуальна машина (клон першої) повинна мати іншу IP-адресу та унікальне ім’я хоста. Це дозволяє легко розрізняти машини під час передачі файлів через SSH. Процес налаштування аналогічний до попереднього кроку, але з іншими значеннями адреси та імені.

```
# Зміна імені хоста:
sudo hostnamectl set-hostname comp2

# Редагування мережевої конфігурації:
sudo nano /etc/netplan/01-netcfg.yaml
```

У файлі вказується IP-адреса, відмінна від першої машини (наприклад, 192.168.2.61):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp0s3:
      dhcp4: no
      addresses: [192.168.2.61/24]
      routes:
        - to: default
          via: 192.168.2.1
```

```
    nameservers:
      addresses: [8.8.8.8, 8.8.4.4]
```

Застосування конфігурації:

```
sudo netplan apply
```

Після цього слід перезавантажити систему, щоб зміни імені хоста набули чинності:

```
sudo reboot
```

Після перезавантаження виконайте hostname, щоб переконатися, що ім’я змінено на comp2.

3.2. Реалізація безпечного обміну даними за допомогою асиметричної криптографії

Ця частина лабораторної роботи моделює повний цикл безпечного обміну даними між двома сторонами за допомогою асиметричної криптографії. Процес починається з генерації ключової пари на стороні отримувача, експорту відкритого ключа та його передачі відправнику. Далі відправник шифрує повідомлення відкритим ключем отримувача, передає зашифрований файл через захищений канал SSH, а отримувач розшифровує його за допомогою свого закритого ключа. Цей сценарій демонструє ключову властивість асиметричної криптографії: дані, зашифровані відкритим ключем, можуть бути розшифровані лише відповідним закритим ключем, що забезпечує конфіденційність навіть при перехопленні трафіку.

3.2.1. Встановлення необхідного програмного забезпечення. Для виконання лабораторної роботи на обох віртуальних машинах має бути встановлено SSH-сервер (для безпечної передачі файлів) та GnuPG (для шифрування). Ці компоненти є основою безпечної комунікації в Linux-середовищі. Після встановлення необхідно переконатися, що служба SSH активна та працює коректно.

```
# Оновлення списків пакетів та встановлення ПЗ:
sudo apt update
sudo apt install openssh-server gnupg

# Перевірка статусу SSH-сервера:
sudo systemctl status ssh
```

Якщо служба не активна, її можна запустити командою sudo systemctl start ssh. У виводі команди status має бути рядок active (running).

3.2.2. Генерація ключової пари на машині отримувача. На другій віртуальній машині (comp2), яка виступає в ролі отримувача, створюється пара ключів: відкритий (public key) та закритий (private key). Закритий ключ залишається на цій машині й ніколи не передається, тоді як відкритий ключ може бути вільно розповсюджений. Під час генерації студент вводить своє ім'я, email та надійний пароль для захисту закритого ключа. Цей пароль знадобиться пізніше для розшифрування даних.

```
# Створіть пару ключів для шифрування:
gpg --gen-key
```

Під час виконання команди слід дотримуватися інструкцій у терміналі: вибрати тип ключа (за замовчуванням — RSA), довжину ключа (рекомендовано 2048 або 3072 біт), термін дії (можна залишити безстроковим) та ввести особисті дані.

Після створення ключової пари експортується відкритий ключ у текстовому форматі (ASCII-armored), що полегшує його передачу:

```
gpg --export --armor your_email@example.com > public_key.asc
```

Замініть *your_email@example.*com на email, вказаний під час генерації ключа. Файл *public_key.asc* містить відкритий ключ у читабельному вигляді.

3.2.3. Передача відкритого ключа на машину відправника. Для того щоб відправник міг зашифрувати повідомлення, йому потрібен відкритий ключ отримувача. Цей ключ передається з машини *comp2* на *comp1* за допомогою протоколу SCP (Secure Copy), який використовує SSH для шифрування трафіку. Це гарантує, що навіть при перехопленні трафіку ключ не буде скомпрометований.

```
# На comp2: визначення IP-адреси comp1
ip a

# Передача ключа на comp1 (замініть 192.168.2.60 на реальну IP
comp1):
scp public_key.asc gns3@192.168.2.60:/home/gns3/
```

Після успішної передачі на машині *comp1* імпортується отриманий відкритий ключ:

```
# На comp1:
gpg --import public_key.asc
```

Для перевірки успішного імпорту виконайте:

```
gpg --list-keys
```

У виводі має з’явитися запис з ім’ям та email, вказаними під час генерації ключа на *comp2*.

3.2.4. Шифрування та передача повідомлення. На машині *comp1* (відправник) створюється простий текстовий файл, який містить довільне повідомлення. Цей файл шифрується за допомогою відкритого ключа отримувача. В результаті створюється бінарний файл з розширенням *.gpg*, який може бути розшифрований лише власником відповідного закритого ключа. Після шифрування файл передається на *comp2* через SCP.

```
# Створення текстового файлу:
nano message.txt
# (введіть будь-який текст, збережіть та вийдіть)

# Шифрування файлу:
gpg --encrypt --recipient your_email@example.com message.txt

# Передача зашифрованого файлу на comp2:
scp message.txt.gpg gns3@192.168.2.61:/home/gns3/
Після виконання цих команд у домашній директорії comp1 з’явиться
файл message.txt.gpg, а на comp2 — такий самий файл після
передачі.
```

3.2.5. Розшифрування повідомлення на стороні отримувача. На машині comp2 виконується розшифрування отриманого файлу за допомогою закритого ключа. Система запитає пароль, введений під час генерації ключової пари. Після успішного введення пароля буде створено розшифрований файл, вміст якого повинен повністю збігатися з оригінальним повідомленням.

```
# Розшифрування файлу:
gpg --decrypt message.txt.gpg > decrypted_message.txt

# Перегляд результату:
cat decrypted_message.txt
```

Якщо вивід команди cat збігається з текстом, введеним у message.txt на comp1, це свідчить про успішне виконання всього циклу шифрування та розшифрування.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 8

Тема: Створення VPN-тунелю IPsec за допомогою StrongSwan

Мета роботи: Набуття практичних навичок у налаштуванні та тестуванні захищеного VPN-з'єднання між двома вузлами за допомогою програмного забезпечення StrongSwan на базі операційної системи Debian 12.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Віртуальні приватні мережі та їх архітектура

Віртуальна приватна мережа (VPN) - це технологія, що забезпечує створення захищеного з'єднання через незахищену мережу, таку як Інтернет. Основна мета VPN полягає у забезпеченні конфіденційності, цілісності та автентифікації передачі даних між різними вузлами або мережами. VPN-з'єднання створюють віртуальний тунель між двома точками, через який передаються зашифровані дані, що робить їх недоступними для перехоплення та аналізу сторонніми особами. Використання VPN дозволяє організаціям забезпечити безпечний віддалений доступ до корпоративних ресурсів, об'єднати географічно розподілені офіси в єдину мережу та захистити конфіденційну інформацію під час її передачі через публічні мережі. Технологія VPN широко застосовується як в корпоративному середовищі для забезпечення безпеки бізнес-комунікацій, так і приватними користувачами для забезпечення анонімності та захисту даних під час роботи в Інтернеті.

В архітектурі VPN можна виділити кілька ключових компонентів: VPN-клієнт, який встановлюється на пристрої користувача; VPN-сервер, що відповідає за прийом з'єднань від клієнтів та маршрутизацію їх до потрібної мережі; тунельні протоколи, які визначають як саме відбувається передача даних; та системи шифрування й автентифікації, що забезпечують захист даних. В залежності від використовуваних протоколів та конфігурацій, VPN може забезпечувати різні рівні безпеки та продуктивності. Важливою характеристикою VPN є здатність інкапсулювати пакети одного протоколу в пакети іншого, що дозволяє створювати зашифрований тунель для передачі даних. Процес шифрування та інкапсуляції реалізується шляхом додавання спеціальних заголовків до початкових пакетів, що забезпечує їх захищену передачу через незахищені мережі.

Існує декілька основних топологій VPN, кожна з яких має свої особливості та сфери застосування:

1 Топологія "точка-точка" (Site-to-Site) використовується для з'єднання двох локальних мереж через Інтернет, створюючи єдину віртуальну мережу. Ця топологія часто використовується для об'єднання географічно розподілених офісів компанії.

2 Топологія "віддалений доступ" (Remote Access) забезпечує безпечне підключення окремих користувачів до корпоративної мережі через Інтернет.

Користувачі встановлюють на своїх пристроях спеціальне програмне забезпечення для підключення до VPN-сервера організації.

3Топологія "DMVPN" (Dynamic Multipoint VPN) дозволяє створювати динамічні з'єднання між багатьма кінцевими точками, забезпечуючи гнучку маршрутизацію та масштабованість мережі.

4 Топологія "Hub-and-Spoke" передбачає наявність центрального вузла (Hub), через який проходить весь трафік між підключеними клієнтами (Spokes), що спрощує управління мережею, але може створювати вузьке місце у вигляді центрального вузла.

У контексті нашої лабораторної роботи ми будемо використовувати топологію "точка-точка", де дві віртуальні машини з операційною системою Debian 12 встановлюватимуть захищений VPN-тунель між собою за допомогою програмного забезпечення StrongSwan. Ця конфігурація дозволить нам дослідити основні принципи роботи VPN, зокрема механізми автентифікації, шифрування та встановлення захищеного з'єднання. Для забезпечення безпеки з'єднання буде використовуватись протокол IPsec, який є стандартом де-факто для побудови захищених VPN-тунелів і реалізується програмним забезпеченням StrongSwan. Такий підхід дозволяє створити надійну і стабільну інфраструктуру для захищеного обміну даними між двома вузлами.

1.2 Протоколи VPN та їх порівняльні характеристики

Існує широкий спектр протоколів для організації VPN-мереж, кожен з яких має свої особливості, переваги та недоліки. Протокол PPTP (Point-to-Point Tunneling Protocol) був одним із перших широко використовуваних VPN-протоколів, розроблений Microsoft для створення віртуальних приватних мереж. PPTP використовує порт TCP 1723 та інкапсуляцію GRE (Generic Routing Encapsulation), забезпечуючи високу швидкість передачі даних, але має суттєві недоліки в системі безпеки, зокрема, уразливості в методах автентифікації та шифрування. L2TP (Layer 2 Tunneling Protocol) часто використовується разом з IPsec для формування L2TP/IPsec, що забезпечує тунелювання на другому рівні моделі OSI з додатковим шифруванням від IPsec. Цей протокол використовує порти UDP 500, 1701 та 4500, забезпечує кращу безпеку, ніж PPTP, але може бути повільнішим та складнішим у налаштуванні. IPsec (Internet Protocol Security) - це набір протоколів для забезпечення безпеки на рівні мережевого протоколу IP, який може використовуватися самостійно або в поєднанні з іншими протоколами. IPsec забезпечує високий рівень безпеки завдяки надійним механізмам шифрування та автентифікації, хоча може бути складним у налаштуванні та вимагати спеціальних знань.

Протокол OpenVPN - це відкрите програмне забезпечення для створення захищених VPN-з'єднань з використанням бібліотеки OpenSSL для шифрування. Цей протокол працює через TCP або UDP на порту 1194 (за замовчуванням, але може бути змінений), забезпечує високий рівень безпеки та гнучкості, хоча і потребує додаткової установки клієнтського програмного забезпечення. WireGuard - це сучасний, швидкий та мінімалістичний VPN-

протокол, що використовує новітні криптографічні алгоритми та спрощену архітектуру. Він працює через UDP, забезпечує високу продуктивність та низьку затримку, дуже простий у налаштуванні, але є відносно новим протоколом, що може обмежувати його використання в деяких середовищах. SSTP (Secure Socket Tunneling Protocol) - розроблений Microsoft протокол, який використовує порт TCP 443 та працює за допомогою SSL/TLS, що дозволяє йому обходити більшість брандмауерів та забезпечувати високий рівень безпеки, хоча він не є кросплатформеним і найкраще підтримується на системах Windows. IKEv2 (Internet Key Exchange version 2) часто використовується з IPsec і забезпечує надійний механізм обміну ключами та автентифікації. Він працює через UDP порти 500 та 4500, забезпечує високу стабільність з'єднання, особливо при зміні мереж, та підтримується багатьма платформами.

Таблиця 8.1 – Порівняльна характеристика протоколів VPN

Протокол	Рівень безпеки	Швидкість	Основні переваги	Основні недоліки
PPTP	Низький	Висока	Простота налаштування, висока швидкість	Низький рівень безпеки, легко блокується
L2TP/IPsec	Високий	Середня	Хороша безпека, широка підтримка	Може блокуватися NAT, повільніший ніж інші
IPsec	Високий	Середня до високої	Стандартизований, високий рівень безпеки	Складність налаштування, проблеми з NAT
OpenVPN	Дуже високий	Середня до високої	Гнучкість, безпека, обхід блокувань	Потребує клієнтського ПЗ, складніший у налаштуванні
WireGuard	Високий	Дуже висока	Висока продуктивність, простота	Відносно новий, менше функцій
SSTP	Високий	Середня	Працює через порт HTTPS, обходить блокування	Менша кросплатформність, зав'язка на Microsoft
IKEv2/IPsec	Високий	Висока	Надійність при зміні мереж, безпека	Може блокуватися в деяких мережах

У контексті нашої лабораторної роботи ми використовуємо протокол IPsec, реалізований за допомогою StrongSwan, оскільки він забезпечує високий рівень безпеки та є широко використовуваним стандартом для корпоративних VPN-рішень. IPsec пропонує гнучкі механізми автентифікації та шифрування, які дозволяють забезпечити надійний захист передачі даних між двома вузлами в нашому лабораторному середовищі. Хоча існують і інші протоколи з подібними характеристиками безпеки, IPsec залишається оптимальним вибором для організації захищеного тунелю точка-точка на базі Debian 12, особливо враховуючи наявність потужної та добре документованої реалізації у вигляді StrongSwan.

1.3 Протокол IPsec

IP Security (IPsec) – це набір протоколів, що забезпечують шифрування, аутентифікацію та захист передачі IP-пакетів, який наразі включає майже 20 пропозицій щодо стандартів і 18 RFC. Специфікація IPsec розробляється робочою групою IP Security Protocol IETF і спочатку містила три алгоритмічно незалежні базові специфікації: "Архітектура безпеки IP", "Автентифікаційний заголовок (AH)" та "Інкапсуляція зашифрованих даних (ESP)" (RFC 1825–1827). Однак у листопаді 1998 року група запропонувала оновлені версії цих стандартів (RFC 2401–2412), які зараз мають статус попередніх, тоді як RFC 1825–1827 вже кілька років вважаються застарілими і не використовуються на практиці. Крім того, існує низка алгоритмічно залежних специфікацій, що базуються на таких протоколах, як MD5, SHA та DES.

Основне призначення IPsec полягає у забезпеченні конфіденційності, цілісності, автентичності та захисту від повторного відтворення (replay protection) для IP-пакетів. На відміну від багатьох інших протоколів безпеки, які працюють на вищих рівнях моделі OSI, IPsec функціонує на мережевому рівні (рівень 3), що дозволяє йому захищати весь IP-трафік незалежно від протоколів вищих рівнів. Це робить IPsec універсальним рішенням для захисту різноманітних мережевих комунікацій, від простого обміну файлами до складних корпоративних застосунків. IPsec має дві основні складові: протокол автентифікації заголовків (Authentication Header, AH), який забезпечує цілісність та автентифікацію даних, та протокол інкапсуляції захищених даних (Encapsulating Security Payload, ESP), який додатково забезпечує конфіденційність шляхом шифрування даних.

IPsec може працювати у двох основних режимах: транспортному та тунельному. У транспортному режимі захищається тільки корисне навантаження (payload) IP-пакета, тоді як оригінальні IP-заголовки залишаються незмінними. Цей режим зазвичай використовується для захисту з'єднань між хостами або для захисту трафіку протоколів вищого рівня. У тунельному режимі, який ми використовуємо в нашій лабораторній роботі, весь оригінальний IP-пакет (заголовок та дані) інкапсулюється та шифрується, створюючи новий IP-пакет з новим заголовком. Цей режим найчастіше використовується для створення VPN-з'єднань між мережами або вузлами. Тунельний режим забезпечує вищий рівень безпеки, оскільки приховує оригінальні адреси відправника та отримувача, що ускладнює аналіз трафіку сторонніми особами.

Структура пакета IPsec залежить від використовуваного протоколу (AH чи ESP) та режиму роботи (транспортний чи тунельний). У тунельному режимі з використанням ESP (який ми застосовуємо в лабораторній роботі) структура пакета включає: новий IP-заголовок, який містить адреси шлюзів IPsec; ESP-заголовок, що містить ідентифікатор безпеки (Security Parameter Index, SPI) та послідовний номер для захисту від атак повторного відтворення; інкапсульований оригінальний IP-пакет (заголовок та дані), який повністю зашифрований; ESP-трейлер, що містить дані для доповнення (padding) та

інформацію про наступний заголовок; та ESP-автентифікатор, який забезпечує цілісність та автентичність всього ESP-пакета. Така структура забезпечує комплексний захист даних під час їх передачі через незахищені мережі.

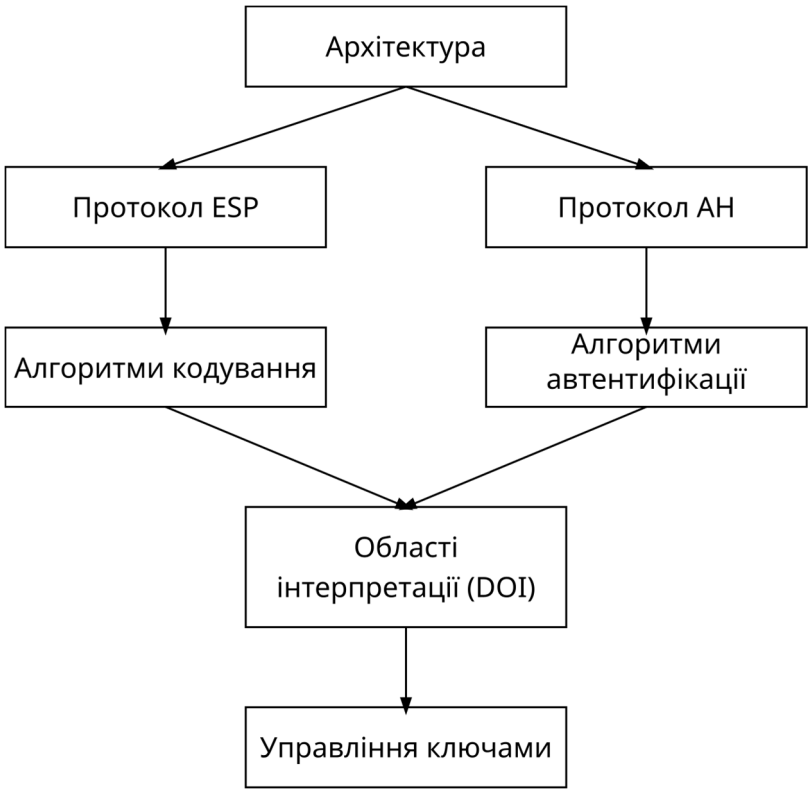

Рисунок 8.1 – Архітектура протоколу IPsec

Для встановлення захищеного з'єднання IPsec використовує протокол обміну ключами IKE (Internet Key Exchange), який забезпечує автентифікацію сторін та узгодження параметрів безпеки. Процес встановлення з'єднання IPsec складається з кількох етапів. Спочатку відбувається ініціалізація IKE-сесії (Phase 1), під час якої сторони автентифікують одна одну та встановлюють захищений канал для подальшого обміну ключами. У нашій лабораторній роботі для цього використовується режим захищеного обміну (main mode) протоколу IKEv1 з PSK (Pre-Shared Key) для автентифікації. На наступному етапі (Phase 2) сторони узгоджують параметри безпеки для IPsec-з'єднання, включаючи алгоритми шифрування, хешування та час життя ключів. Після успішного завершення цих фаз створюється захищене IPsec-з'єднання, через яке можна передавати дані. StrongSwan періодично оновлює ключі відповідно до налаштованих значень часу життя для забезпечення вищого рівня безпеки.

У нашій лабораторній роботі для шифрування даних використовується алгоритм AES256, який забезпечує високий рівень захисту від несанкціонованого доступу до інформації. Для забезпечення цілісності даних застосовується хеш-алгоритм SHA1, який, хоча і не є найновішим, але залишається достатньо надійним для навчальних цілей. Для обміну ключами використовується алгоритм Diffie-Hellman групи modp1024, який забезпечує безпечний обмін ключами через незахищений канал. Ці параметри визначені у

конфігураційному файлі ipsec.conf у рядках "ike=aes256-sha1-modp1024!" та "esp=aes256-sha1!", де символ "!" означає, що інші алгоритми не приймаються. Такий підхід забезпечує високий рівень безпеки VPN-з'єднання, достатній для більшості практичних застосувань.

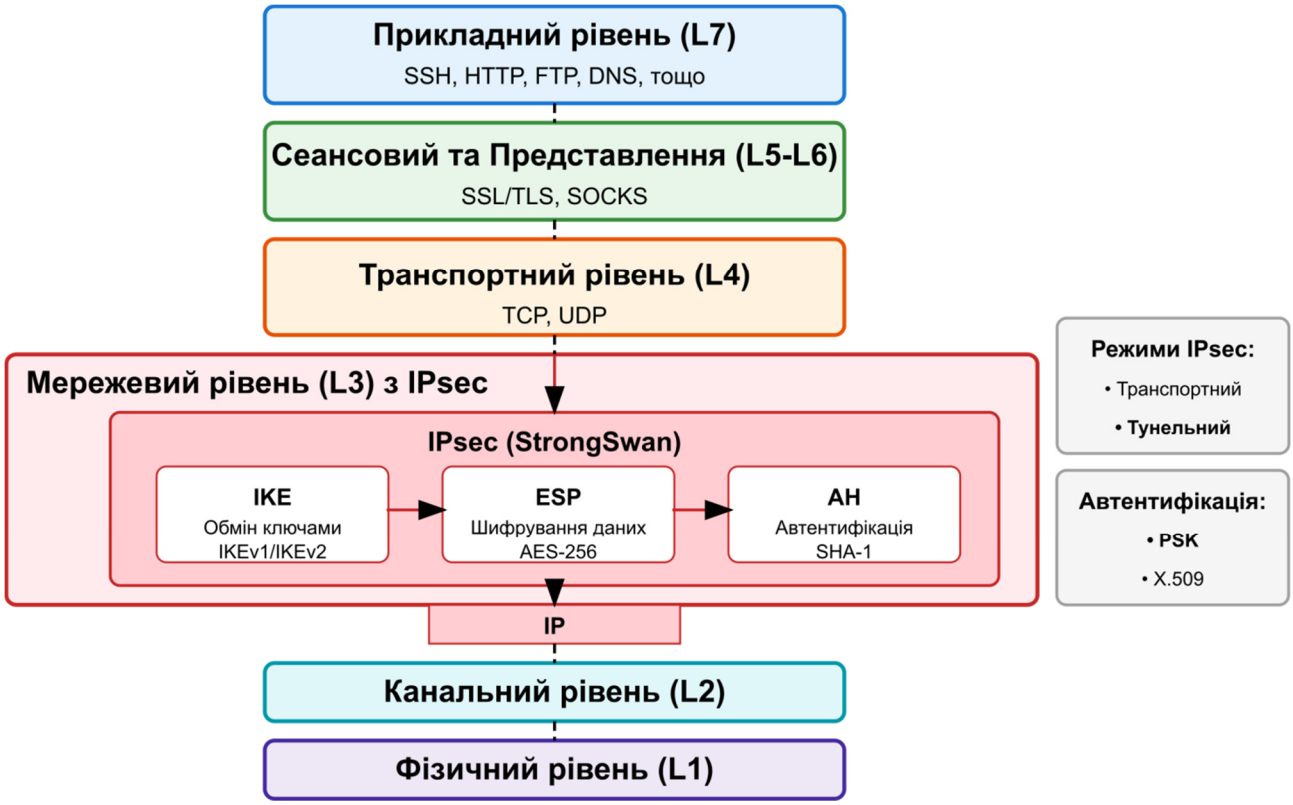

Рисунок 8.2 – Архітектура IPsec у мережевому стеку OSI з використанням StrongSwan

У центральній частині схеми розташовано детальне представлення мережевого рівня з вбудованим у нього набором протоколів IPsec, реалізованих у програмному забезпеченні StrongSwan. Ключовими компонентами цього набору є протокол IKE (Internet Key Exchange), що відповідає за безпечний обмін криптографічними ключами та встановлення захищеного з'єднання між учасниками VPN-тунелю. У лабораторній роботі використовується версія IKEv1, хоча сучасні реалізації StrongSwan підтримують також більш захищену і функціональну версію IKEv2. Протокол IKE функціонує в два етапи: спочатку відбувається автентифікація сторін та створення захищеного каналу для обміну ключами (фаза 1), а потім за допомогою цього каналу відбувається узгодження параметрів захисту для самого IPsec-з'єднання (фаза 2). Важливим елементом налаштування IKE є вибір методу автентифікації, в даній роботі використовується Pre-Shared Key (PSK) — заздалегідь погоджений спільний секретний ключ, хоча в промислових системах перевага часто надається сертифікатам X.509 через їх вищу стійкість до різних видів атак.

Схема також демонструє два основні протоколи захисту даних у складі IPsec: ESP (Encapsulating Security Payload) та AH (Authentication Header). Протокол ESP, що активно використовується в лабораторній роботі, забезпечує

конфіденційність переданих даних шляхом їх шифрування з використанням алгоритму AES-256, який вважається одним із найбільш стійких симетричних шифрів. Окрім шифрування, ESP також може забезпечувати цілісність даних та їх автентифікацію, використовуючи алгоритми хешування, такі як SHA-1, хоча в сучасних реалізаціях рекомендується використовувати більш захищені алгоритми, наприклад, SHA-256. Протокол AH, у свою чергу, фокусується виключно на забезпеченні цілісності даних та автентифікації їх джерела, без шифрування самих даних, що робить його менш поширеним у сучасних VPN-рішеннях, особливо коли потрібно забезпечити високий рівень конфіденційності переданої інформації. На практиці часто використовується лише ESP, який може забезпечити як конфіденційність, так і цілісність даних.

На бокових панелях схеми наведено додаткову інформацію щодо режимів роботи IPsec та методів автентифікації. IPsec може функціонувати у двох режимах: транспортному, коли шифрується лише корисне навантаження IP-пакетів, та тунельному, коли шифрується весь IP-пакет, включно з заголовками, що забезпечує більш високий рівень захисту та приховування внутрішньої топології мережі. У лабораторній роботі використовується саме тунельний режим, про що свідчить відповідне налаштування у конфігураційному файлі (/etc/ipsec.conf). Щодо методів автентифікації, окрім використаного в роботі PSK, схема також згадує сертифікати X.509, які є стандартом для інфраструктури відкритих ключів (PKI) і забезпечують більш масштабоване та безпечне рішення для автентифікації учасників VPN. Взаємозв'язки між компонентами схеми підкреслюються стрілками, що демонструють послідовність обробки даних та взаємозалежність різних протоколів у загальній структурі IPsec.

Візуальне представлення ієрархії протоколів дозволяє краще зрозуміти, як IPsec інтегрується в загальний стек мережевих протоколів і яку роль відіграє кожен з його компонентів у забезпеченні захищеного зв'язку. Над мережевим рівнем з IPsec на схемі розташовані транспортний рівень (L4) з протоколами TCP та UDP, що відповідають за надійну та швидку доставку даних між прикладними програмами, рівні сеансу та представлення (L5-L6), що забезпечують управління сеансами зв'язку та перетворення даних у формати, зрозумілі для прикладних програм, і, нарешті, прикладний рівень (L7), на якому функціонують такі прикладні протоколи, як HTTP, FTP, SSH тощо. Нижче мережевого рівня знаходяться канальний рівень (L2), що забезпечує доставку кадрів між пристроями в локальній мережі, та фізичний рівень (L1), що відповідає за передачу бітів через фізичне середовище. Вертикальні зв'язки між рівнями демонструють, як дані проходять через усі рівні стеку при передачі від одного комп'ютера до іншого, при цьому на кожному рівні вони інкапсулюються у відповідні протокольні одиниці даних (PDU) з додаванням службової інформації, необхідної для коректної обробки на відповідному рівні стеку протоколів.

1.4 Налаштування та використання StrongSwan для організації VPN-тунелю

StrongSwan - це потужна та гнучка реалізація протоколу IPsec з відкритим вихідним кодом, яка широко використовується для створення захищених VPN-з'єднань у середовищах на базі Linux. У нашій лабораторній роботі StrongSwan використовується для встановлення та налаштування захищеного VPN-тунелю між двома віртуальними машинами на базі Debian 12. Налаштування StrongSwan здійснюється за допомогою редагування декількох конфігураційних файлів, які визначають параметри з'єднання, методи автентифікації, алгоритми шифрування та інші важливі аспекти функціонування VPN. Основними конфігураційними файлами StrongSwan є */etc/ipsec.conf*, який містить загальні налаштування та параметри з'єднань, та /etc/ipsec.secrets, який містить секретні ключі та паролі для автентифікації. У деяких випадках також можуть використовуватися файли конфігурації у директорії */etc/strongswan.d/* для більш детального налаштування окремих компонентів системи.

Таблиця 8.2 – Розшифровка параметрів файлу /etc/ipsec.conf

Параметр	Значення	Розшифровка
Загальна секція config setup		
charondebug	"all"	Вмикає детальне логування для всіх компонентів StrongSwan (корисно для налагодження).
uniqueids	yes	Забороняє повторне використання ідентифікаторів сесій для запобігання атак.
Секція з'єднання conn myvpn		
auto	add	З'єднання завантажується, але не активується автоматично (потрібен ручний запуск ipsec up myvpn).
authby	secret	Використовується спільний ключ (PSK) з файлу /etc/ipsec.secrets.
type	tunnel	Режим тунелю (шифрується весь трафік, включаючи IP-заголовки).
keyexchange	ikev1	Використовується протокол IKEv1 для обміну ключами.
left	192.168.2.60	Ліва кінцева точка (поточний сервер).
right	192.168.2.61	Права кінцева точка (віддалений сервер).
ike	aes256-sha1-modp1024!	Алгоритми для фази 1 (IKE): AES-256, SHA-1, DH group 1024. Символ ! забороняє слабкі варіанти.
esp	aes256-sha1!	Алгоритми для фази 2 (ESP): AES-256, SHA-1. Шифрування даних у тунелі.
aggressive	no	Вимикає агресивний режим IKE (менш безпечний).
keyingtries	%forever	Необмежена кількість спроб встановлення ключів.
ikelifetime	28800s (8 год)	Час життя ключів IKE.
lifetime	3600s (1 год)	Час життя ключів IPSec.
dpddelay	30s	Інтервал перевірки зв'язку (Dead Peer Detection).
dpdtimeout	120s	Час очікування відповіді перед вваженням з'єднання мертвим.
dpdaction	restart	Дія при втраті зв'язку: перезапуск з'єднання.

Файл */etc/ipsec.secrets* містить секретні ключі для автентифікації VPN-з'єднань. У нашому випадку він містить рядок:

```
192.168.2.60 192.168.2.61 : PSK 'myvpnpassword'
```

Він визначає спільний ключ (PSK) для автентифікації з'єднання між вузлами з IP-адресами 192.168.2.60 та 192.168.2.61. Такий підхід до автентифікації є відносно простим, але достатньо надійним для багатьох сценаріїв використання VPN. Для підвищення безпеки у виробничих середовищах часто використовуються більш складні методи автентифікації, такі як цифрові сертифікати (X.509) або інтеграція з зовнішніми системами автентифікації. StrongSwan підтримує різноманітні методи автентифікації, включаючи EAP (Extensible Authentication Protocol) з різними варіантами (EAP-TLS, EAP-TTLS, EAP-MSCHAPv2 тощо), автентифікацію на основі RADIUS-сервера, а також різні методи управління ключами.

Схема лабораторного макета для нашої роботи включає дві віртуальні машини з операційною системою Debian 12, з'єднані через віртуальну мережу VirtualBox. Перша машина має IP-адресу 192.168.2.60, друга - 192.168.2.61. На обох машинах встановлено та налаштовано SSH-сервер для забезпечення віддаленого доступу та програмне забезпечення StrongSwan для організації VPN-тунелю. Основним елементом конфігурації є файли */etc/ipsec.conf* та */etc/ipsec.secrets*, які налаштовані для створення захищеного тунелю між двома вузлами з використанням протоколу IPsec у тунельному режимі. Після запуску служби StrongSwan та активації VPN-з'єднання за допомогою команди "ipsec up myvpn" між машинами встановлюється захищений канал зв'язку, через який передаються зашифровані дані. Для перевірки стану з'єднання використовується команда "ipsec status", яка відображає інформацію про активні з'єднання та їх параметри. Також можна налаштувати автоматичний запуск VPN-тунелю при старті системи, змінивши параметр "auto=add" на "auto=start" у файлі конфігурації.

1.5 Схема лабораторного стенду

Лабораторний стенд складається з двох віртуальних машин на базі Debian 12, з'єднаних віртуальною мережею у середовищі VirtualBox. Перша віртуальна машина має IP-адресу 192.168.2.60, друга - 192.168.2.61. Обидві машини мають встановлений SSH-сервер для віддаленого доступу та програмне забезпечення StrongSwan для організації VPN-тунелю. Віртуальні машини об'єднані у спільну мережу, яка імітує незахищене середовище передачі даних. На обох машинах налаштовані служби IPsec з відповідними конфігураційними файлами: /etc/ipsec.conf містить параметри VPN-з'єднання, а */etc/ipsec.secrets* зберігає спільний ключ для автентифікації. Демон charon на кожній машині відповідає за обробку IKE-протоколу та підтримку IPsec-з'єднань. Між машинами встановлюється захищений IPsec-тунель у режимі tunnel з використанням шифрування AES-256 та хешування SHA-1. Трафік між віртуальними

машинами передається в зашифрованому вигляді, забезпечуючи конфіденційність та цілісність даних.

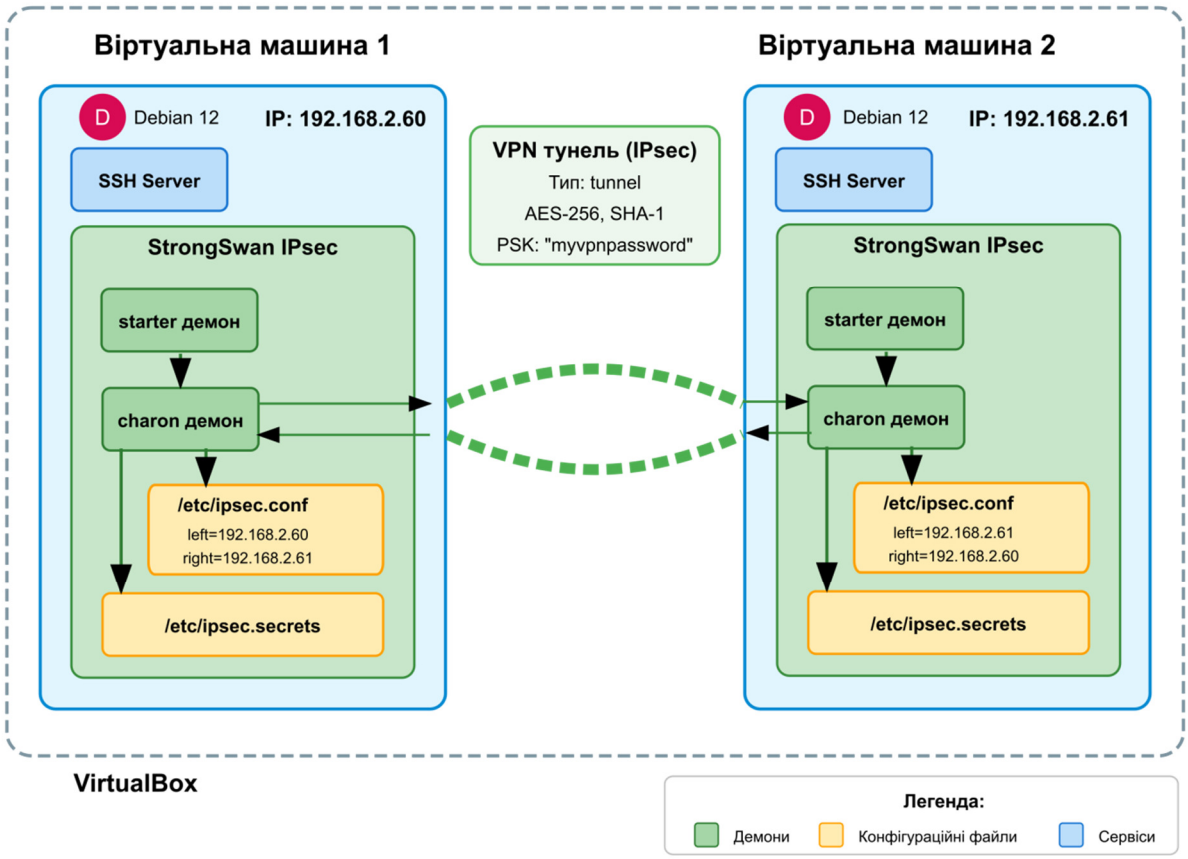

Рисунок 8.3 – Архітектура VPN-тунелю IPsec за допомогою StrongSwan між двома вузлами

Таблиця 8.3 – Налаштування ВМ в прикладі виконання лабораторного завдання

Параметр	Віртуальна машина 1	Віртуальна машина 2
IP-адреса	192.168.2.60	192.168.2.61
Операційна система	Debian 12	Debian 12
Роль в VPN-тунелі	Ініціатор/Респондент	Респондент/Ініціатор
Основні служби	StrongSwan, SSH	StrongSwan, SSH
Конфігураційні файли	/etc/ipsec.conf, /etc/ipsec.secrets	/etc/ipsec.conf, /etc/ipsec.secrets
Метод автентифікації	Pre-Shared Key (PSK)	Pre-Shared Key (PSK)
Алгоритм шифрування	AES-256	AES-256
Алгоритм хешування	SHA-1	SHA-1
Група Діффі-Хеллмана	modp1024	modp1024

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Налаштування першої віртуальної машини (ініціатора тунелю)

У цій частині виконується повна підготовка першої віртуальної машини (з IP-адресою 192.168.2.60), яка буде виступати в ролі ініціатора VPN-тунелю

на основі протоколу IPsec. Процес включає встановлення програмного забезпечення StrongSwan — потужної реалізації IPsec для Linux, налаштування основного конфігураційного файлу */etc/ipsec.conf*, у якому визначаються параметри з'єднання, алгоритми шифрування, режими роботи та адреси кінцевих точок, а також створення файлу */etc/ipsec.secrets*, що містить спільний ключ автентифікації (Pre-Shared Key, PSK). Ці дії формують базову конфігурацію для безпечного обміну даними за протоколом IPsec у тунельному режимі, де весь IP-трафік між двома вузлами буде повністю інкапсульований та зашифрований. Особливу увагу слід приділити синтаксису конфігураційних файлів, оскільки навіть одна помилка може призвести до невдалого встановлення тунелю.

3.1.1. Встановлення та активація служби StrongSwan. Для організації VPN-тунелю на основі протоколу IPsec у середовищі Debian 12 використовується програмний пакет StrongSwan — потужна реалізація IPsec з відкритим кодом, яка підтримує широкий спектр алгоритмів шифрування, автентифікації та обміну ключами, а також надійну інтеграцію з ядром Linux через демон charon. Першим етапом є встановлення цього пакета з офіційного репозиторію системи. Після встановлення необхідно активувати службу strongswan-starter, яка відповідає за запуск демона charon — основного компонента StrongSwan, що обробляє IKE-протокол (Internet Key Exchange) та управління IPsec-асоціаціями. Успішний запуск служби є обов'язковою умовою для подальшої роботи з IPsec, оскільки саме charon відповідає за встановлення захищеного з'єднання, генерацію ключів та підтримку тунелю.

```
sudo apt update
sudo apt install strongswan
```

Після встановлення слід запустити службу та переконатися, що вона працює коректно:

```
sudo systemctl start strongswan-starter
sudo systemctl status strongswan-starter
```

У виводі команди *status* має бути рядок *active (running*). Якщо служба не запущена, перевірте наявність помилок у журналах (*journalctl -u strongswan-starter*) та переконайтеся, що система має доступ до Інтернету для завантаження залежностей. Також важливо зазначити, що після встановлення StrongSwan автоматично створює базові конфігураційні файли, але вони потребують ручного редагування для відповідності нашому сценарію «точка-точка».

3.1.2 Налаштування основного конфігураційного файлу IPsec

Основні параметри VPN-з'єднання визначаються у файлі */etc/ipsec.conf*. У цьому файлі задаються глобальні налаштування (секція *config setup*) та

параметри конкретного з'єднання (секція *conn myvpn*). Для нашого сценарію використовується тунельний режим (type = tunnel), протокол IKEv1 (keyexchange = ikev1), алгоритми AES-256 для шифрування та SHA-1 для хешування, а також механізм Dead Peer Detection (DPD) для контролю живості з'єднання. Адреса *left* вказує на локальну машину (ініціатор), а *right* — на віддалену (респондент). Параметр *auto=add* означає, що з'єднання буде завантажено, але не активовано автоматично — його потрібно запускати вручну командою *ipsec up*. Такий підхід дає більший контроль над процесом встановлення тунелю, що важливо в навчальному середовищі.

```
sudo nano /etc/ipsec.conf
```

У файл додається наступний вміст:

```
config setup
    charondebug="all"
    uniqueids=yes

conn myvpn
    auto=add
    authby=secret
    type=tunnel
    keyexchange=ikev1
    left=192.168.2.60
    right=192.168.2.61
    ike=aes256-sha1-modp1024!
    esp=aes256-sha1!
    aggressive=no
    keyingtries=%forever
    ikelifetime=28800s
    lifetime=3600s
    dpddelay=30s
    dpdtimeout=120s
    dpdaction=restart
```

Після збереження файлу рекомендується перевірити його синтаксис. Хоча команда *ipsec checkconf* не завжди доступна, головне — уникнути синтаксичних помилок, таких як відсутність відступів, неправильні лапки або пропущені символи. Зверніть увагу на символ ! наприкінці рядків *ike* та *esp* — він забороняє використання слабших альтернативних алгоритмів, що підвищує рівень безпеки.

3.1.3. Налаштування спільного ключа автентифікації. Для автентифікації сторін у нашому сценарії використовується Pre-Shared Key (PSK) — секретний пароль, який має бути однаковим на обох вузлах. Цей ключ зберігається у файлі */etc/ipsec.secrets*, який за замовчуванням доступний лише користувачу root. У цьому файлі вказується пара IP-адрес, для якої діє ключ, та сам ключ у лапках. Важливо, щоб ключ був достатньо складним (наприклад,

myvpnpassword — лише для навчальних цілей; у продакшні використовуйте довгі випадкові паролі). Цей файл є критичним з точки зору безпеки, тому його права доступу мають бути обмежені.

```
sudo nano /etc/ipsec.secrets
```

У файл додається наступний рядок:

```
192.168.2.60 192.168.2.61 : PSK "myvpnpassword"
```

Після збереження файлу переконайтеся, що права доступу встановлені правильно:

```
sudo chmod 600 /etc/ipsec.secrets
sudo chown root:root /etc/ipsec.secrets
```

Це запобігає несанкціонованому читанню секретного ключа. Якщо права будуть надто відкритими (наприклад, 644), StrongSwan відмовиться запускатися з помилкою безпеки.

3.2 Налаштування другої віртуальної машини (респондента тунелю)

Друга віртуальна машина (з IP-адресою 192.168.2.61) виступає в ролі респондента, тобто очікує на вхідні запити на встановлення VPN-тунелю. Її конфігурація майже ідентична до першої, але з однією ключовою відмінністю: параметри left та right міняються місцями, щоб відображати роль цієї машини як «лівої» (локальної) у своєму контексті. Такий підхід забезпечує симетричну конфігурацію, необхідну для двонаправленого з’єднання. Обидві машини повинні мати однаковий PSK, інакше автентифікація не відбудеться.

3.2.1. Встановлення StrongSwan та налаштування конфігурації. Виконайте ті самі дії, що й на першій машині: встановіть StrongSwan, створіть файл */etc/ipsec.conf*, але з іншими значеннями *left* та *right*.

```
sudo apt update
sudo apt install strongswan
sudo systemctl start strongswan-starter
```

Налаштуйте /etc/ipsec.conf:

```
sudo nano /etc/ipsec.conf
```

Вміст файлу:

```
config setup
    charondebug="all"
```

```
    uniqueids=yes

conn myvpn
    auto=add
    authby=secret
    type=tunnel
    keyexchange=ikev1
    left=192.168.2.61
    right=192.168.2.60
    ike=aes256-sha1-modp1024!
    esp=aes256-sha1!
    aggressive=no
    keyingtries=%forever
    ikelifetime=28800s
    lifetime=3600s
    dpddelay=30s
    dpdtimeout=120s
    dpdaction=restart
```

Аналогічно налаштуйте */etc/ipsec.secrets*:

```
sudo nano /etc/ipsec.secrets
```

Відредагуйте його зміст:

```
192.168.2.61 192.168.2.60 : PSK "myvpnpassword"
```

Не забудьте встановити правильні права:

```
sudo chmod 600 /etc/ipsec.secrets
```

Після цього переконайтеся, що служба запущена: *sudo systemctl status strongswan-starter*. Якщо все налаштовано правильно, обидві машини готові до встановлення тунелю.

3.3 Активація та перевірка VPN-тунелю. На цьому етапі виконується запуск VPN-тунелю, його активація та верифікація стану. Спочатку перезапускається служба StrongSwan на обох машинах, щоб застосувати нові конфігурації. Потім на одній із машин (наприклад, на ініціаторі) виконується команда *ipsec up myvpn*, яка ініціює процес встановлення IKE-сесії та IPsec-тунелю. Після успішного встановлення стан з’єднання можна перевірити за допомогою команди *ipsec status*. Опціонально можна налаштувати автоматичний запуск тунелю під час завантаження системи.

3.3.1 Застосування конфігурації та запуск тунелю. На обох машинах перезапустіть службу StrongSwan, щоб завантажити оновлені конфігураційні файли:

```
sudo systemctl restart strongswan-starter
```

Ініціюйте встановлення тунелю з першої машини:

```
sudo ipsec up myvpn
```

Якщо все налаштовано правильно, у терміналі з’явиться повідомлення про успішне встановлення CHILD_SA (IPsec-тунелю). У разі помилки (наприклад, «no matching peer config found») перевірте, чи співпадають IP-адреси в *left/right* та чи однаковий PSK.

3.3.2 Перевірка стану з’єднання. Для підтвердження того, що тунель активний, виконайте:

```
sudo ipsec status
```

У виводі має бути рядок ESTABLISHED, а також інформація про використані алгоритми, SPI (Security Parameter Index) та тривалість сесії. Наприклад:

```
myvpn{1}:  ESTABLISHED  10 seconds ago, ...
```

Це означає, що тунель працює. Якщо з’єднання не встановлено, перевірте: - чи співпадають PSK на обох машинах, - чи правильно вказані IP-адреси, - чи немає блокування портів UDP 500/4500 фаєрволом (у навчальному середовищі фаєрвол зазвичай вимкнено).

Також можна переглянути журнали демона charon:

```
sudo journalctl -u strongswan-starter -f
```

Це допоможе зрозуміти, на якому етапі (IKE Phase 1 чи Phase 2) сталася помилка.

3.3.3. Налаштування автоматичного запуску (опціонально). Щоб тунель запускався автоматично після перезавантаження, змініть параметр *auto=add* на *auto=start* у файлі */etc/ipsec.conf* на обох машинах:

```
conn myvpn
    auto=start
    ...
```

Після цього перезавантажте систему:

```
sudo reboot
```

Після завантаження перевірте стан:

```
sudo ipsec status
```

```
gns3@comp1:~$ sudo ipsec status
Security Associations (1 up, 0 connecting):
       myvpn[1]: ESTABLISHED 118 minutes ago, 192.168.2.60[192.168.2.60]...192.168.2.61[192.168.2.61]
       myvpn{3}:  REKEYED, TUNNEL, reqid 1, expires in 27 minutes
       myvpn{3}:   192.168.2.60/32 === 192.168.2.61/32
       myvpn{4}:  INSTALLED, TUNNEL, reqid 1, ESP SPIs: c62e13e0_i c7083ee5_o
       myvpn{4}:   192.168.2.60/32 === 192.168.2.61/32
gns3@comp1:~$
```

Рисунок 8.4 – Перевірка роботи VPN-тунелю в терміналі

Тунель має бути активним без ручного втручання. Це важливо для продакшн-середовищ, де стабільність з'єднання критична.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов'язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема,

помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 9

Тема: Налаштування захищених протоколів передачі даних SSH та TLS

Мета роботи: Ознайомитися з принципами побудови захищених протоколів передачі даних, навчитися налаштовувати SSH та TLS/SSL у середовищі Debian Linux, проаналізувати процеси автентифікації, обміну ключами та виявити типові помилки конфігурації.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Архітектура та принципи роботи протоколу SSH

Сучасні інформаційні системи функціонують у розподіленому середовищі, де обмін даними між віддаленими вузлами є невід'ємною частиною робочих процесів. Віддалене адміністрування серверів, передача конфіденційної інформації через публічні мережі, доступ до веб-додатків – усі ці операції вимагають надійного захисту від несанкціонованого перехоплення, модифікації чи підробки даних. Саме тому протоколи захищеної передачі даних стали критично важливим компонентом сучасної інфраструктури інформаційної безпеки. До найпоширеніших захищених протоколів належать SSH (Secure Shell) для віддаленого керування системами та TLS/SSL (Transport Layer Security / Secure Sockets Layer) для захисту веб-трафіку та інших мережевих з'єднань. Обидва протоколи базуються на криптографічних механізмах, які забезпечують три фундаментальні властивості безпеки: конфіденційність (шифрування даних), цілісність (захист від модифікації) та автентифікацію (підтвердження справжності сторін комунікації).

Протокол SSH був розроблений у 1995 році фінським дослідником Тату Юльоненом як відповідь на вразливості традиційних протоколів віддаленого доступу, таких як Telnet, rlogin та FTP, які передавали дані, включно з паролями, у відкритому вигляді. SSH забезпечує повне шифрування сесії, що робить неможливим перехоплення облікових даних навіть у незахищених мережах. Сьогодні SSH є стандартом де-факто для адміністрування Unix-подібних систем і широко використовується для автоматизації, тунелювання трафіку та безпечної передачі файлів. Протокол TLS, який є наступником SSL, забезпечує захист на транспортному рівні моделі OSI і використовується в HTTPS, електронній пошті, VoIP та багатьох інших додатках. Основною відмінністю TLS від SSH є його призначення: якщо SSH орієнтований на інтерактивні сесії та керування системами, то TLS створений для захисту будь-якого TCP-трафіку, особливо веб-комунікацій між браузерами та серверами.

Протокол SSH побудований за модульною архітектурою, що складається з трьох основних рівнів: транспортного, автентифікаційного та рівня з'єднання. Транспортний рівень відповідає за встановлення безпечного каналу зв'язку між клієнтом і сервером, виконуючи обмін криптографічними ключами, вибір алгоритмів шифрування та ініціалізацію захищеної сесії. Процес обміну

ключами базується на алгоритмах Діффі-Геллмана або його еліптичних варіантах, таких як ECDH (Elliptic Curve Diffie-Hellman) або Curve25519, які дозволяють двом сторонам сформувати спільний секретний ключ через незахищений канал без попереднього обміну секретною інформацією. Після встановлення сесійного ключа весь трафік шифрується симетричними алгоритмами, такими як AES (Advanced Encryption Standard) у режимах GCM або CTR, або ChaCha20-Poly1305, що забезпечує високу швидкість обробки даних при збереженні криптографічної стійкості.

Автентифікаційний рівень SSH підтримує декілька методів перевірки справжності користувачів, серед яких найпоширенішими є автентифікація на основі паролів та асиметричних криптографічних ключів. Парольна автентифікація, хоча й проста у впровадженні, вважається менш безпечною через вразливість до атак підбору (brute-force) та соціальної інженерії. Автентифікація за ключами використовує пари відкритий-закритий ключ, де відкритий ключ зберігається на сервері у файлі authorized_keys, а закритий залишається у користувача. Під час автентифікації сервер надсилає випадковий виклик, який клієнт підписує своїм закритим ключем, і сервер перевіряє підпис за допомогою відкритого ключа. Цей механізм забезпечує значно вищий рівень безпеки, оскільки закритий ключ ніколи не передається мережею і може додатково захищатися парольною фразою. Сучасні реалізації SSH підтримують алгоритми цифрового підпису RSA, DSA, ECDSA та Ed25519, причому останній набуває популярності завдяки високій швидкості та компактності ключів при збереженні високого рівня безпеки.

Рівень з’єднання SSH дозволяє мультиплексувати декілька логічних каналів через єдине захищене з’єднання, що використовується для інтерактивних сесій оболонки, виконання окремих команд, переадресації портів та передачі файлів. Кожен канал ідентифікується унікальним номером і може мати власні параметри керування потоком даних. Механізм переадресації портів (port forwarding) дозволяє створювати захищені тунелі для незахищених протоколів, пропускаючи їхній трафік через SSH-з’єднання. Локальна переадресація (local forwarding) дозволяє клієнту підключатися до віддалених сервісів через SSH-сервер, віддалена переадресація (remote forwarding) надає зворотну функціональність, а динамічна переадресація (dynamic forwarding) створює SOCKS-проксі для маршрутизації довільного TCP-трафіку. Ці можливості роблять SSH універсальним інструментом не лише для адміністрування, але й для побудови складних мережевих топологій та обходу мережевих обмежень.

Конфігурація SSH-сервера здійснюється через файл sshd_config, який містить численні параметри безпеки та функціональності. Критично важливими налаштуваннями є вибір дозволених методів автентифікації, обмеження користувачів, які мають право підключатися, налаштування портів прослуховування та визначення переліку дозволених криптографічних алгоритмів. Параметри PubkeyAuthentication та PasswordAuthentication визначають, які методи автентифікації будуть прийматися сервером, причому рекомендована практика передбачає увімкнення автентифікації за ключами та

вимкнення парольної для критичних систем. Параметр PermitRootLogin контролює можливість прямого входу з правами суперкористувача, і його значення should be set to “prohibit-password” або “no” для запобігання атакам на адміністративний обліковий запис. Налаштування Ciphers, KexAlgorithms та MACs дозволяють явно вказати, які криптографічні алгоритми може використовувати сервер, що дає змогу вимкнути застарілі та вразливі варіанти, забезпечуючи відповідність сучасним стандартам безпеки.

У таблиці 9.1 наведено основні команди для роботи з SSH-сервером у системах на базі Debian Linux, які охоплюють встановлення, налаштування, моніторинг та діагностику служби.

Таблиця 9.1 – Команди управління SSH-сервером

Команда	Призначення
sudo apt install openssh-server	Встановлення пакету OpenSSH-сервера
sudo systemctl start ssh	Запуск служби SSH
sudo systemctl stop ssh	Зупинка служби SSH
sudo systemctl restart ssh	Перезапуск служби SSH (застосування змін конфігурації)
sudo systemctl enable ssh	Увімкнення автоматичного запуску SSH при завантаженні системи
sudo systemctl status ssh	Перевірка статусу служби SSH
sudo nano /etc/ssh/sshd_config	Редагування конфігураційного файлу SSH-сервера
ssh -Q cipher	Перегляд підтримуваних алгоритмів шифрування
ssh -Q kex	Перегляд підтримуваних алгоритмів обміну ключами
ssh -Q mac	Перегляд підтримуваних алгоритмів контролю цілісності
sudo grep "sshd" /var/log/auth.log	Перегляд журналу подій SSH-автентифікації
ssh-keygen -t ed25519	Генерація пари ключів Ed25519 для автентифікації
ssh-copy-id user@host	Копіювання відкритого ключа на віддалений сервер

Журналювання подій SSH є важливим механізмом аудиту безпеки, що дозволяє відстежувати всі спроби підключення, успішні та невдалі автентифікації, а також аномальну активність. У системах на базі Debian журнали SSH зберігаються у файлі /var/log/auth.log, який містить записи про всі події автентифікації та авторизації в системі. Аналіз цього журналу дозволяє виявити атаки підбору паролів за характерними записами множинних невдалих спроб входу з одного IP-адреси, спроби підключення до неіснуючих облікових записів або підозрілу активність у незвичний час доби. Для автоматизації моніторингу можуть використовуватися спеціалізовані інструменти, такі як fail2ban, які аналізують журнали в реальному часі та автоматично блокують IP-адреси, що демонструють підозрілу поведінку, створюючи правила файрволу після перевищення порогового значення невдалих спроб автентифікації.

1.2 Протокол TLS/SSL та інфраструктура відкритих ключів

Протокол TLS є криптографічним протоколом, призначеним для забезпечення безпечної комунікації через комп’ютерні мережі, і найчастіше використовується для захисту HTTP-трафіку, утворюючи протокол HTTPS.

TLS працює між транспортним та прикладним рівнями моделі OSI, забезпечуючи прозорий захист для додатків вищого рівня без необхідності модифікації їхньої логіки. Протокол еволюціонував від SSL версій 1.0, 2.0 та 3.0 (розроблених компанією Netscape у 1990-х роках) до TLS 1.0, 1.1, 1.2 та найновішої версії TLS 1.3, прийнятої у 2018 році. Сучасні стандарти безпеки рекомендують використовувати виключно TLS 1.2 та 1.3, оскільки попередні версії містять відомі вразливості та застарілі криптографічні механізми. TLS 1.3 суттєво покращує продуктивність та безпеку за рахунок скорочення кількості обмінів даними під час встановлення з'єднання (handshake) та видалення небезпечних або застарілих криптографічних наборів.

Процес встановлення TLS-з'єднання (TLS handshake) є складною послідовністю обмінів, що включає узгодження версії протоколу, вибір криптографічних алгоритмів, обмін сертифікатами та генерацію сесійних ключів. Процес починається з надсилання клієнтом повідомлення ClientHello, яке містить підтримувані версії TLS, список криптографічних наборів (cipher suites) та випадкове число для генерації ключів. Сервер відповідає повідомленням ServerHello, вибираючи версію протоколу та набір шифрів із запропонованих клієнтом, після чого надсилає свій X.509-сертифікат для автентифікації. Клієнт перевіряє сертифікат сервера, звіряючи ланцюг довіри до кореневого центру сертифікації, перевіряючи терміни дії та валідність цифрового підпису. Після успішної верифікації обидві сторони виконують обмін ключами за допомогою алгоритму Діффі-Геллмана або RSA, формуючи спільний секрет, з якого виводяться симетричні ключі для шифрування подальшого трафіку.

Інфраструктура відкритих ключів (Public Key Infrastructure, PKI) є критичним компонентом екосистеми TLS, що забезпечує механізм довіри та автентифікації у відкритих мережах. PKI базується на ієрархії центрів сертифікації (Certificate Authorities, CA), які виступають довіреними третіми сторонами, що підтверджують справжність відкритих ключів шляхом підписання цифрових сертифікатів. Кореневі CA мають самопідписані сертифікати, які вбудовані в операційні системи та браузери як довірені анкери (trust anchors). Проміжні CA отримують сертифікати від кореневих CA і можуть видавати сертифікати кінцевим сутностям, створюючи ланцюг довіри. Коли клієнт отримує сертифікат сервера, він перевіряє весь ланцюг до кореневого CA, гарантуючи, що кожен сертифікат у ланцюгу підписаний приватним ключем вищестоящого CA і що жоден із сертифікатів не відкликаний через компрометацію або інші причини.

Сертифікат X.509 є стандартизованою структурою даних, що містить відкритий ключ сутності разом із ідентифікаційною інформацією та цифровим підписом центру сертифікації. Основні поля сертифіката включають версію формату X.509, серійний номер (унікальний для кожного CA), алгоритм підпису, ім'я емітента (CA), період дії (not before та not after), ім'я суб'єкта (Subject Distinguished Name, що ідентифікує власника сертифіката), відкритий ключ та його параметри, а також розширення, які додають додаткову функціональність. Поле Subject Alternative Name (SAN) є критично важливим

розширенням для веб-сертифікатів, оскільки дозволяє вказати декілька доменних імен, для яких дійсний сертифікат, включаючи підстановочні символи для піддоменів. Розширення Key Usage та Extended Key Usage визначають дозволені способи використання ключа, наприклад, для підпису сертифікатів, шифрування даних або автентифікації серверів, що допомагає обмежити потенційний збиток від компрометації ключа.

Самопідписані сертифікати є X.509-сертифікатами, в яких емітент і суб'єкт співпадають, тобто сертифікат підписаний власним приватним ключем власника без залучення зовнішнього центру сертифікації. Такі сертифікати широко використовуються у тестових середовищах, внутрішніх корпоративних мережах та для швидкого розгортання HTTPS без витрат на комерційні сертифікати, однак вони не забезпечують довіри з боку зовнішніх клієнтів. Браузери та інші клієнтські додатки видають попередження безпеки при зустрічі з самопідписаними сертифікатами, оскільки не можуть перевірити справжність сервера через відсутність довіреного ланцюга до кореневого CA. Це робить самопідписані сертифікати непридатними для публічних веб-сервісів, де важлива довіра користувачів, але цілком прийнятними для внутрішніх систем, де адміністратори можуть вручну додати сертифікати до списку довірених або де попередження браузера можна ігнорувати з розумінням контексту.

У таблиці 9.2 представлено команди для роботи з TLS/SSL-сертифікатами та налаштування веб-сервера Apache з підтримкою HTTPS, які охоплюють генерацію сертифікатів, конфігурацію сервера та діагностику з'єднань.

Таблиця 9.2 – Команди роботи з TLS/SSL та Apache

Команда	Призначення
sudo apt install apache2	Встановлення веб-сервера Apache
sudo a2enmod ssl	Увімкнення модуля SSL в Apache
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /path/key.key -out /path/cert.crt	Генерація самопідписаного сертифіката з приватним ключем RSA 2048 біт
sudo openssl req -x509 -nodes -days 365 -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout /path/key.key -out /path/cert.crt	Генерація самопідписаного сертифіката з еліптичним ключем ECDSA
sudo nano /etc/apache2/sites-available/default-ssl.conf	Редагування конфігурації HTTPS-сайту Apache
sudo a2ensite default-ssl	Увімкнення HTTPS-сайту в Apache
sudo systemctl reload apache2	Перезавантаження конфігурації Apache без переривання з'єднань
openssl s_client -connect hostname:443 -servername hostname	Аналіз TLS-з'єднання та деталей сертифіката
openssl x509 -in cert.crt -text -noout	Перегляд деталей сертифіката у текстовому форматі
openssl verify -CAfile ca-bundle.crt cert.crt	Перевірка валідності сертифіката проти ланцюга CA

Налаштування Apache для роботи з HTTPS вимагає активації модуля mod_ssl та створення віртуального хосту, який слухає на порту 443 (стандартний порт HTTPS) та використовує сертифікат для автентифікації. Конфігураційний файл віртуального хосту повинен містити директиви SSLEngine для увімкнення SSL/TLS, SSLCertificateFile для вказівки шляху до файлу сертифіката та SSLCertificateKeyFile для вказівки приватного ключа. Додаткові параметри безпеки включають SSLProtocol для визначення дозволених версій TLS (рекомендовано “all -SSLv3 -TLSv1 -TLSv1.1” для використання лише TLS 1.2 та 1.3), SSLCipherSuite для вказівки дозволених криптографічних наборів з пріоритетом сучасних алгоритмів, таких як ECDHE-RSA-AES256-GCM-SHA384, та SSLHonorCipherOrder для надання пріоритету переліку шифрів сервера над переліком клієнта. Конфігурація може також включати директиву Header для примусового використання HTTPS через механізм HSTS (HTTP Strict Transport Security), який інструктує браузери завжди підключатися до сайту через HTTPS, навіть якщо користувач ввів URL з HTTP.

Інструмент openssl s_client є потужним засобом діагностики TLS-з’єднань, що дозволяє встановити з’єднання з TLS-сервером та отримати детальну інформацію про всі етапи handshake, використані криптографічні алгоритми та сертифікати. Команда виводить повний ланцюг сертифікатів, починаючи від сертифіката сервера до кореневого CA, дозволяючи перевірити його цілісність та валідність. Параметр -servername використовується для передачі індикатора імені сервера (Server Name Indication, SNI), що є критичним для серверів, які хостять декілька HTTPS-сайтів на одній IP-адресі. У виводі команди міститься інформація про код повернення верифікації (verify return code), де код 0 означає успішну перевірку ланцюга довіри, код 18 вказує на самопідписаний сертифікат, а інші коди сигналізують про різні проблеми, такі як закінчення терміну дії, невідповідність імені хоста або відкликання сертифіката. Аналіз цієї інформації дозволяє діагностувати проблеми з конфігурацією сертифікатів до того, як вони вплинуть на кінцевих користувачів.

Рисунок 9.1 ілюструє структуру процесу TLS handshake між клієнтом та сервером, демонструючи послідовність обміну повідомленнями та формування захищеного каналу.

На діаграмі зображені дві вертикальні лінії, що представляють клієнта (ліворуч) та сервер (праворуч). Між ними показані горизонтальні стрілки, що відображають обмін повідомленнями у хронологічному порядку зверху вниз. Перша стрілка йде від клієнта до сервера і позначена як “ClientHello” із вказівкою підтримуваних версій TLS та cipher suites. Друга стрілка йде від сервера до клієнта і позначена як “ServerHello + Certificate + ServerKeyExchange + ServerHelloDone”, що представляє відповідь сервера з вибраними параметрами та сертифікатом. Третя стрілка від клієнта до сервера позначена як “ClientKeyExchange + ChangeCipherSpec + Finished”, що означає завершення обміну ключами клієнтом. Четверта стрілка від сервера до клієнта позначена як “ChangeCipherSpec + Finished”, що підтверджує готовність сервера до

шифрованої комунікації. Після цього показані двосторонні стрілки з підписом “Encrypted Application Data”, що символізують захищений обмін даними додатка. Діаграма наочно демонструє, що handshake складається з чотирьох основних етапів перед початком передачі захищених даних.

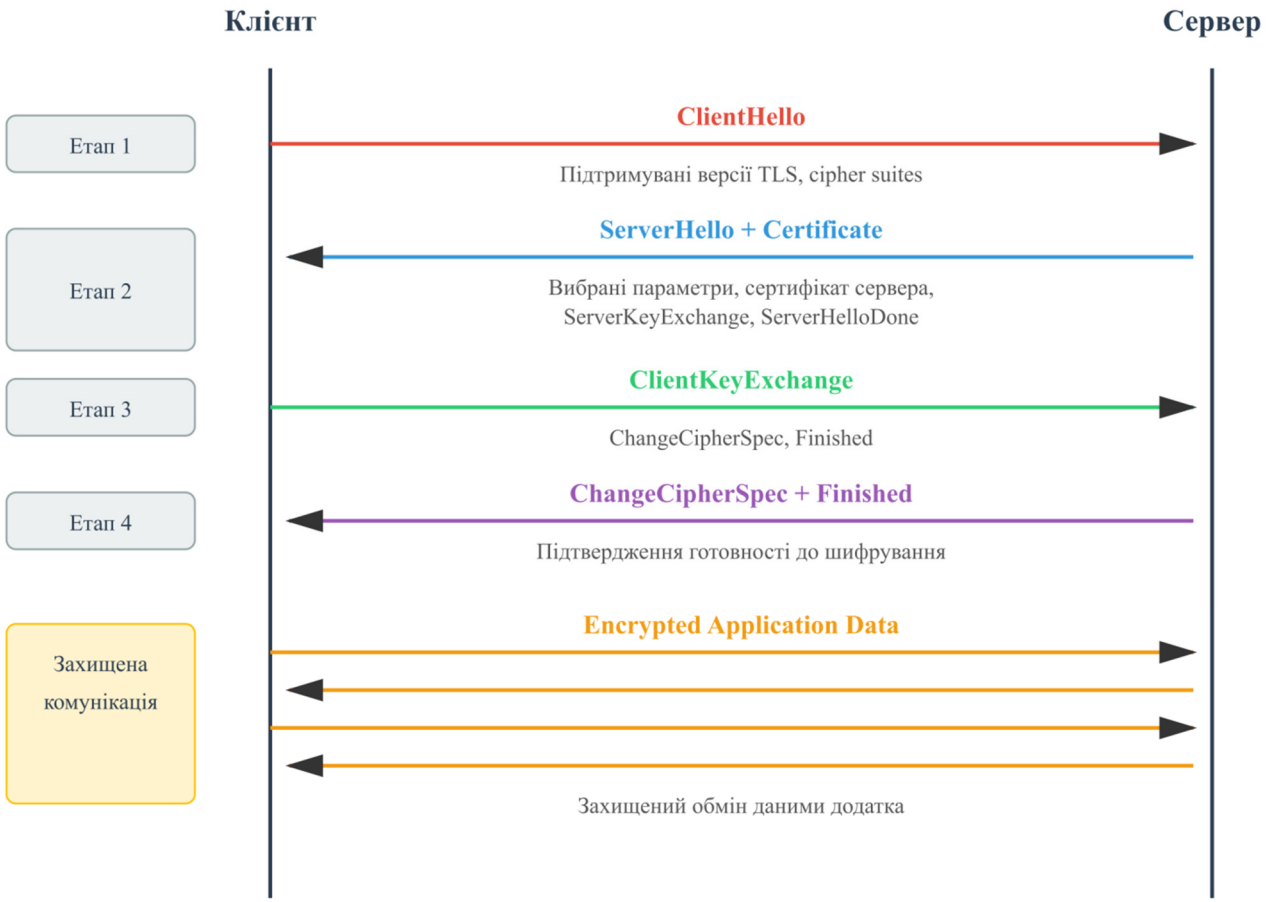

Рисунок 9.1 – Діаграма процесу TLS handshake

1.3 Типові помилки конфігурації та їх наслідки

Неправильна конфігурація SSH-серверів є однією з найпоширеніших причин компрометації систем, оскільки SSH часто є єдиною точкою віддаленого доступу до критичної інфраструктури. Вимкнення автентифікації на основі ключів та залишення лише парольної автентифікації значно знижує рівень безпеки, оскільки робить сервер вразливим до атак грубої сили, де зловмисники автоматично перебирають тисячі паролів у спробі вгадати вірну комбінацію. Навіть складні паролі можуть бути скомпрометовані при достатньому часі та обчислювальних ресурсах, особливо якщо використовуються словникові слова або передбачувані комбінації. Автентифікація за ключами усуває цю вразливість, оскільки приватний ключ довжиною 2048 або 4096 біт для RSA чи 256 біт для Ed25519 практично неможливо підібрати навіть з використанням сучасних суперкомп’ютерів протягом розумного часу. Додатково, приватний ключ може бути захищений

парольною фразою, створюючи двофакторну автентифікацію (щось, що має користувач — ключ, і щось, що знає — пароль до ключа).

Дозвіл прямого входу під обліковим записом root є критичною помилкою безпеки, яка надає зловмисникам пряму мету для атак з максимальними привілеями. Якщо вдається скомпрометувати обліковий запис root, атакуючий отримує повний контроль над системою без необхідності подальшої ескалації привілеїв. Краща практика передбачає вимкнення прямого входу root через SSH (встановлення PermitRootLogin no або prohibit-password) та вимагання від адміністраторів спочатку входити під звичайними обліковими записами, а потім використовувати sudo для виконання привілейованих операцій. Це створює додатковий рівень захисту та аудиту, оскільки всі дії, виконані через sudo, логуються з ідентифікацією конкретного користувача. Використання застарілих криптографічних алгоритмів, таких як 3DES, RC4, CBC-режимів для шифрування або MD5 та SHA1 для хешування, робить з'єднання вразливим до відомих атак, таких як BEAST, POODLE чи Sweet32, що дозволяють зловмисникам дешифрувати трафік або підробляти повідомлення.

У контексті TLS/SSL найпоширенішими помилками є використання самопідписаних сертифікатів у виробничих середовищах, застарілих версій протоколу та слабких криптографічних наборів. Самопідписані сертифікати, хоча й забезпечують шифрування трафіку, не надають гарантій справжності сервера, що робить користувачів вразливими до атак "людина посередині" (man-in-the-middle, MITM). Атакуючий може перехопити з'єднання, надати власний самопідписаний сертифікат та розшифрувати весь трафік між клієнтом і сервером, при цьому користувач побачить те саме попередження про недовірений сертифікат, що й при легітимному з'єднанні. У виробничих системах критично важливо використовувати сертифікати від довірених комерційних CA або безкоштовних сервісів, таких як Let's Encrypt, які забезпечують автоматизовану видачу та оновлення сертифікатів з періодом дії 90 днів, стимулюючи регулярну ротацію.

Відсутність належної перевірки сертифікатів на стороні клієнта є іншою критичною вразливістю, коли додатки програмно вимикають валідацію сертифікатів для "спрощення" розробки або тестування. Це повністю нівелює захист TLS, оскільки дозволяє будь-якому сертифікату бути прийнятим, навіть якщо він самопідписаний, закінчився термін дії або виданий для іншого доменного імені. Розробники іноді залишають такий код у виробничих версіях додатків, створюючи серйозну вразливість безпеки. Використання застарілих версій TLS, таких як SSL 3.0, TLS 1.0 або TLS 1.1, експонує систему до відомих атак, включаючи POODLE (Padding Oracle On Downgraded Legacy Encryption), BEAST (Browser Exploit Against SSL/TLS) та інших, які експлуатують криптографічні слабкості цих протоколів. Сучасні стандарти безпеки вимагають підтримки виключно TLS 1.2 та TLS 1.3, причому TLS 1.3 є найбільш бажаним завдяки вдосконаленому handshake, видаленню небезпечних функцій та обов'язковому використанню forward secrecy.

Неправильне налаштування cipher suites може призвести до використання слабких алгоритмів шифрування або відсутності forward secrecy, властивості,

яка гарантує, що компрометація довгострокових ключів сервера не дозволить дешифрувати раніше перехоплений трафік. Forward secrecy досягається використанням ефемерних ключів у алгоритмах обміну ключами Діффі-Геллмана (DHE або ECDHE), де для кожної сесії генерується новий тимчасовий ключ, який знищується після завершення з'єднання. Набори шифрів, що використовують RSA для обміну ключами (наприклад, TLS_RSA_WITH_AES_128_CBC_SHA), не забезпечують forward secrecy, оскільки використовують статичний приватний ключ сервера для дешифрування pre-master secret, надісланого клієнтом. Якщо цей ключ буде скомпрометований у майбутньому, зловмисник зможе дешифрувати весь збережений трафік, зашифрований за його допомогою. Пріоритет повинен надаватися наборам шифрів з ECDHE (еліптичний Діффі-Геллман з ефемерними ключами) та автентифікацією на основі RSA або ECDSA, наприклад ECDHE-RSA-AES256-GCM-SHA384 або ECDHE-ECDSA-CHACHA20-POLY1305.

1.4 Моніторинг та аудит безпеки захищених з'єднань

Ефективний моніторинг SSH-активності є критичним компонентом оборонної стратегії, що дозволяє виявляти атаки на ранніх стадіях та швидко реагувати на інциденти безпеки. Регулярний аналіз журналів автентифікації дозволяє ідентифікувати підозрілі патерни, такі як множинні невдалі спроби входу з одного джерела, що може вказувати на автоматизовану атаку підбору паролів, або успішні входи з незвичайних географічних локацій чи в нетиповий час доби. Інструменти автоматизованого моніторингу, такі як fail2ban, logwatch або OSSEC, можуть аналізувати журнали в реальному часі та виконувати автоматичні дії у відповідь на виявлені загрози. Fail2ban працює за принципом динамічного блокування IP-адрес, які демонструють зловмисну поведінку, шляхом створення тимчасових правил у файрволі після перевищення налаштованого порогу невдалих спроб автентифікації протягом визначеного часового вікна.

Конфігурація fail2ban для захисту SSH передбачає створення jail (ізольованого набору правил моніторингу), який відстежує файл /var/log/auth.log на предмет записів про невдалі спроби входу через SSH. Типова конфігурація може блокувати IP-адресу на 10 хвилин після 5 невдалих спроб протягом 10-хвилинного вікна, що є балансом між захистом від атак та мінімізацією впливу на легітимних користувачів, які можуть помилитися з паролем. Більш складні сценарії можуть включати прогресивне збільшення тривалості блокування для повторних порушників або постійне блокування після певної кількості циклів бан-розбан. Централізований збір логів з множини серверів у SIEM-систему (Security Information and Event Management) дозволяє виявляти розподілені атаки, коли зловмисник використовує різні IP-адреси для атак на різні сервери, що може бути непомітно при аналізі журналів окремого сервера, але стає очевидним при агрегованому аналізі.

Для TLS/SSL-захищених сервісів моніторинг включає відстеження термінів дії сертифікатів, аналіз використовуваних версій протоколів та cipher suites, а також виявлення аномальних патернів з'єднань. Сертифікати мають обмежений термін дії, і їх несвоєчасне оновлення призводить до переривання обслуговування та попереджень безпеки для користувачів. Автоматизовані інструменти моніторингу можуть періодично перевіряти сертифікати на серверах та надсилати сповіщення адміністраторам за кілька тижнів до закінчення терміну дії, надаючи достатньо часу для оновлення. Let's Encrypt та інші ACME-сумісні CA надають автоматизовані інструменти, такі як certbot, які можуть автоматично оновлювати сертифікати без втручання адміністратора, усуваючи людський фактор із процесу управління сертифікатами.

Сканування SSL/TLS-конфігурації серверів за допомогою спеціалізованих інструментів, таких як SSL Labs' SSL Server Test, testssl.sh або nmap з відповідними скриптами, дозволяє виявити вразливості конфігурації до того, як вони будуть експлуатовані зловмисниками. Ці інструменти перевіряють підтримку застарілих версій протоколів, наявність слабких cipher suites, правильність налаштування HSTS, підтримку forward secrecy та вразливість до відомих атак, таких як Heartbleed, POODLE, FREAK або Logjam. Результати сканування надаються у вигляді детального звіту з оцінкою (часто від A+ до F) та конкретними рекомендаціями щодо покращення конфігурації. Регулярне сканування (наприклад, щомісячне або після кожної зміни конфігурації) повинно бути частиною процесу управління безпекою для забезпечення відповідності найкращим практикам.

У таблиці 9.3 наведено інструменти та методи аудиту безпеки SSH та TLS/SSL, які використовуються для виявлення вразливостей та моніторингу відповідності стандартам безпеки.

Таблиця 9.3 – Інструменти аудиту безпеки SSH та TLS/SSL

Інструмент	Призначення
fail2ban	Автоматичне блокування IP-адрес після виявлення підозрілої активності в журналах
ssh-audit	Аналіз криптографічної конфігурації SSH-сервера та виявлення слабких алгоритмів
nmap --script ssh2-enum-algos	Перелік підтримуваних SSH-алгоритмів віддаленого сервера
testssl.sh	Комплексне сканування TLS/SSL-конфігурації з виявленням вразливостей
openssl s_client -connect host:443 -tls1_2	Тестування підтримки конкретної версії TLS
sslscan	Швидке сканування SSL/TLS-сервера для виявлення підтримуваних шифрів та протоколів
nmap --script ssl-enum-ciphers -p 443 host	Перелік cipher suites та їх оцінка безпеки
SSL Labs SSL Server Test	Онлайн-сервіс для комплексного аналізу HTTPS-конфігурації з детальним звітом
logwatch	Автоматизований аналіз системних журналів з генерацією звітів
certbot renew --dry-run	Тестування процесу оновлення сертифікатів Let's Encrypt

Практика безпечного управління приватними ключами є фундаментальною для збереження цілісності криптографічного захисту. Приватні ключі повинні зберігатися з мінімальними правами доступу (зазвичай 600 або 400 у Unix-системах), доступними лише власнику, і ніколи не повинні передаватися незахищеними каналами або зберігатися у публічно доступних місцях. Для SSH-ключів рекомендується використовувати парольні фрази для додаткового захисту приватного ключа, щоб навіть у разі його копіювання зловмисником, він не міг бути використаний без знання пароля. Для серверних TLS-сертифікатів приватні ключі повинні зберігатися на захищених файлових системах з обмеженим доступом, а у високобезпечних середовищах можуть використовуватися апаратні модулі безпеки (Hardware Security Modules, HSM), які зберігають ключі у спеціалізованому обладнанні та виконують криптографічні операції без експортування ключів назовні.

1.5 Сучасні тенденції та перспективи розвитку

Розвиток квантових обчислень створює нову загрозу для сучасних криптографічних систем, оскільки квантові алгоритми, такі як алгоритм Шора, теоретично здатні ефективно розв’язувати задачі факторизації великих чисел та дискретного логарифму, на яких базуються RSA та алгоритми Діффі-Геллмана. Це означає, що після появи достатньо потужних квантових комп’ютерів більшість існуючих криптографічних протоколів стануть вразливими до дешифрування. У відповідь на цю загрозу активно розробляються постквантові криптографічні алгоритми (Post-Quantum Cryptography, PQC), які мають залишатися стійкими навіть проти квантових атак. Національний інститут стандартів та технологій США (NIST) проводить процес стандартизації PQC-алгоритмів, і у 2024 році були оголошені перші стандартизовані алгоритми, включаючи CRYSTALS-Kyber для інкапсуляції ключів та CRYSTALS-Dilithium для цифрових підписів, які базуються на проблемах решіток у багатовимірних просторах.

Інтеграція постквантової криптографії у SSH та TLS вже розпочалася на експериментальному рівні, причому деякі реалізації вже підтримують гібридні режими, де використовуються як класичні, так і постквантові алгоритми одночасно для забезпечення захисту проти обох типів загроз. Гібридний підхід є перехідною стратегією, яка забезпечує безпеку навіть якщо один із алгоритмів виявиться скомпрометованим, оскільки зловмисник повинен буде зламати обидва алгоритми для отримання доступу до даних. OpenSSH версії 9.0 та новіші включають експериментальну підтримку постквантових алгоритмів обміну ключами, а робочі групи IETF активно працюють над стандартизацією PQC-розширень для TLS 1.3. Перехід на постквантову криптографію буде поступовим процесом, що займе роки, але організації вже зараз повинні планувати міграційні стратегії та оцінювати готовність своєї інфраструктури до майбутніх змін.

Автоматизація управління сертифікатами через протокол ACME (Automatic Certificate Management Environment) революціонізувала екосистему PKI, зробивши процес отримання, оновлення та відкликання сертифікатів повністю автоматизованим. Let’s Encrypt, безкоштовний центр сертифікації, що використовує ACME, видав понад мільярд сертифікатів і значно сприяв масовому впровадженню HTTPS в Інтернеті. Автоматизація усуває проблеми забутих оновлень та дорогих процесів ручного управління, дозволяючи використовувати короткі терміни дії сертифікатів (90 днів для Let’s Encrypt), що зменшує вікно можливостей для зловмисників у разі компрометації ключа. Інструменти, такі як certbot, acme.sh та інтегровані в веб-сервери модулі автоматизації ACME, робять процес оновлення прозорим для адміністраторів, перетворюючи управління сертифікатами з джерела потенційних проблем на рутинну фонову операцію.

Концепція нульової довіри (Zero Trust) стає домінуючою парадигмою у сучасній архітектурі безпеки, де жоден вузол або з’єднання не вважається довіреним за замовчуванням, навіть якщо воно знаходиться у внутрішній корпоративній мережі. У контексті SSH та TLS це означає обов’язкову автентифікацію та шифрування для всіх з’єднань, використання багатофакторної автентифікації, короткострокових сертифікатів з динамічною видачею на основі поточного контексту та ідентичності, а також постійний моніторинг та валідацію всіх активних сесій. SSH-сертифікати (на відміну від традиційних authorized_keys) дозволяють централізоване управління доступом з обмеженими термінами дії та можливістю прив’язки до конкретних хостів або команд, що відповідає принципам Zero Trust. Взаємна TLS-автентифікація (mutual TLS або mTLS), де не лише сервер автентифікується перед клієнтом, але й клієнт надає свій сертифікат серверу, стає стандартом для міжсервісної комунікації у мікросервісних архітектурах та service mesh рішеннях, таких як Istio або Linkerd.

Впровадження механізмів прозорості сертифікатів (Certificate Transparency, CT) значно покращило безпеку екосистеми PKI шляхом створення публічних, лише для додавання логів усіх виданих сертифікатів. Центри сертифікації зобов’язані публікувати всі видані сертифікати у CT-логи, що дозволяє власникам доменів моніторити видачу сертифікатів для їхніх доменів та швидко виявляти несанкціоновану або шахрайську видачу. Браузери, такі як Google Chrome, вимагають наявності Signed Certificate Timestamps (SCT) у сертифікатах як доказ реєстрації у CT-логах, відмовляючись довіряти сертифікатам без них. Інструменти моніторингу CT-логів дозволяють організаціям налаштувати автоматичні сповіщення про видачу нових сертифікатів для їхніх доменів, створюючи систему раннього попередження про потенційні атаки з використанням підроблених сертифікатів. Ця прозорість створює додатковий рівень підзвітності для центрів сертифікації та значно ускладнює проведення атак “людина посередині” з використанням зловмисно виданих сертифікатів.

2 КЛЮЧОВІ ПИТАННЯ

1. У чому полягає основна мета протоколів SSH та TLS/SSL, і як вони реалізують три ключові властивості безпеки — конфіденційність, цілісність та автентифікацію?
2. Які криптографічні алгоритми використовує SSH для обміну ключами, шифрування та контролю цілісності, і чому деякі з них (наприклад, arcfour, diffie-hellman-group1-sha1) вважаються небезпечними?
3. Чим відрізняється автентифікація за ключами від парольної автентифікації в SSH, і як саме відбувається процес перевірки справжності користувача при використанні асиметричних ключів?
4. Які наслідки для безпеки системи має вимкнення автентифікації за ключами (PubkeyAuthentication no) і залишення лише парольної автентифікації у конфігурації SSH-сервера?
5. Що таке X.509-сертифікат, які основні поля він містить, і як він використовується в процесі TLS handshake для автентифікації сервера?
6. У чому принципова різниця між самопідписаним сертифікатом і сертифікатом, виданим довіреним центром сертифікації (CA), і чому перший не забезпечує довіри в публічних середовищах?
7. Як інструмент openssl s_client допомагає аналізувати TLS-з’єднання, і як за його виводом визначити, чи був успішно верифікований ланцюг довіри до кореневого CA?
8. Що означає код помилки Verify return code: 18 у виводі openssl s_client, і як він відрізняється від коду 0?
9. Які типові помилки конфігурації SSH та TLS/SSL найчастіше знижують рівень захисту системи, і як їх можна уникнути?
10. Яким чином журнали подій (/var/log/auth.log) та команди аналізу (ssh -Q, openssl s_client) допомагають адміністратору оцінювати стан безпеки захищених протоколів у Linux-системі?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Налаштування та аналіз безпеки SSH-сервера

3.1.1. Встановлення та базове налаштування OpenSSH-сервера. Протокол SSH (Secure Shell) є фундаментальним інструментом для захищеного віддаленого доступу до серверів у сучасних IT-середовищах. Він забезпечує конфіденційність, цілісність та автентифікацію за рахунок криптографічного шифрування всього трафіку між клієнтом і сервером. У цьому кроці студент встановлює та активує SSH-сервер на основі пакета openssh-server у Debian Linux. Це дозволяє створити базову інфраструктуру для подальшого аналізу алгоритмів шифрування, журналів подій та механізмів автентифікації. Активація служби через systemctl enable --now гарантує, що сервер буде запускатися автоматично після кожного перезавантаження системи, що є стандартною практикою для серверних середовищ.

```
sudo apt update && sudo apt install -y openssh-server
sudo systemctl enable --now ssh
sudo systemctl status ssh
```

Після виконання команд слід переконатися, що служба ssh активна (active (running)) і слухає порт 22. Якщо служба не запускається, слід перевірити наявність конфліктів (наприклад, інший процес уже використовує порт 22) або помилки у конфігураційному файлі /etc/ssh/sshd_config. Успішний статус підтверджує, що сервер готовий приймати вхідні з'єднання.

3.1.2. Аналіз підтримуваних алгоритмів шифрування. SSH використовує різні криптографічні алгоритми для обміну ключами (KEX), шифрування даних (cipher) та контролю цілісності (MAC). Деякі з них, такі як ssh-rsa без SHA-2 або arcfour, вважаються застарілими та вразливими. Для аналізу підтримуваних алгоритмів використовується команда ssh -Q, яка виводить список усіх механізмів, доступних у встановленій версії OpenSSH. Це дозволяє оцінити криптографічну стійкість сервера та виявити потенційні слабкі місця, які можуть бути використані зловмисниками для зниження рівня захисту.

```
ssh -Q cipher       # алгоритми шифрування
ssh -Q kex          # алгоритми обміну ключами
ssh -Q mac          # алгоритми контролю цілісності
```

Студент повинен звернути увагу на наявність сучасних алгоритмів, таких як chacha20-poly1305@openssh.com, aes256-gcm@openssh.com, curve25519-sha256, і відсутність небезпечних (наприклад, arcfour, diffie-hellman-group1-sha1). Якщо виявлено застарілі алгоритми, їх можна вимкнути шляхом редагування /etc/ssh/sshd_config, що є важливим етапом підвищення безпеки.

3.1.3. Перевірка журналів автентифікації. Усі спроби підключення до SSH-сервера реєструються у системному журналі /var/log/auth.log. Це дозволяє адміністратору відстежувати як успішні, так і невдалі спроби входу, що є критичним для виявлення підозрілої активності. Для аналізу журналів використовується команда grep з фільтрацією за ключовим словом sshd. У виводі можна побачити записи типу Accepted publickey (успішна автентифікація за ключем), Failed password (невдала спроба з паролем) або Connection closed (перерване з'єднання).

```
sudo grep "sshd" /var/log/auth.log
```

Цей крок формує розуміння того, як система фіксує події безпеки, і підготовлює до наступного етапу — моделювання сценарію з вимкненою автентифікацією за ключами.

3.1.4. Тестування з вимкненою автентифікацією на основі ключів.

Для демонстрації типової помилки конфігурації студент тимчасово вимикає автентифікацію за ключами, залишаючи лише парольну. Це робиться шляхом редагування файлу */etc/ssh/sshd_config*: параметр *PubkeyAuthentication* встановлюється в *no*, а *PasswordAuthentication* — у (yes). Після перезапуску служби студент намагається підключитися з використанням SSH-ключа. Система має відмовити у використанні ключа і запропонувати ввести пароль. Цей сценарій ілюструє, як неправильна конфігурація значно знижує рівень безпеки, оскільки парольна автентифікація є вразливою до атак грубої сили.

```
sudo nano /etc/ssh/sshd_config
# Додати або змінити рядки:
PubkeyAuthentication no
PasswordAuthentication yes

sudo systemctl restart ssh
ssh user@localhost
```

Після тестування рекомендується повернути оригінальні налаштування (PubkeyAuthentication yes) для збереження безпеки системи.

3.2 Налаштування та аналіз TLS/SSL-захисту веб-сервера

3.2.1. Створення самопідписаного X.509-сертифіката. Протокол TLS/SSL забезпечує захист веб-трафіку, гарантуючи конфіденційність та автентичність даних. Для початку налаштування HTTPS необхідно створити X.509-сертифікат. У навчальних цілях використовується самопідписаний сертифікат, який генерується за допомогою інструменту openssl. Команда створює приватний ключ (RSA, 2048 біт) та сертифікат, дійсний 365 днів. Під час виконання буде запитано інформацію про країну, організацію та Common Name (CN). Останній має відповідати доменному імені або IP-адресі сервера (наприклад, localhost), щоб уникнути попереджень у клієнті.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/ssl/private/apache-selfsigned.key \
  -out /etc/ssl/certs/apache-selfsigned.crt
```

Хоча самопідписаний сертифікат не надає довіри з боку клієнтів, він дозволяє налаштувати HTTPS і проаналізувати механізми шифрування та обміну ключами.

3.2.2 Налаштування веб-сервера Apache з підтримкою HTTPS. Apache — один із найпоширеніших веб-серверів, який легко інтегрується з TLS через модуль *mod_ssl*. У цьому кроці студент встановлює Apache, активує модуль SSL і налаштовує віртуальний хост для HTTPS. Конфігураційний файл */etc/apache2/sites-available/default-ssl.conf* містить шляхи до сертифіката та

приватного ключа. Активація сайту через a2ensite і перезавантаження служби гарантує, що сервер буде прослуховувати порт 443 і використовувати наданий сертифікат.

```
sudo apt install -y apache2
sudo a2enmod ssl
sudo systemctl restart apache2

sudo nano /etc/apache2/sites-available/default-ssl.conf
# Переконатися, що вказано:
SSLEngine on
SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

sudo a2ensite default-ssl
sudo systemctl reload apache2
```

Після цього веб-сервер має бути доступний за адресою *https://<IP_адреса>*.

3.2.3. Аналіз TLS-з'єднання за допомогою openssl s_client. Інструмент openssl s_client дозволяє встановити TLS-з'єднання вручну та проаналізувати деталі сертифіката, алгоритми шифрування та ланцюг довіри. У випадку самопідписаного сертифіката клієнт отримає попередження Verify return code: 18 (self signed certificate), що свідчить про відсутність довіри. Однак саме з'єднання встановлюється, і весь трафік шифрується. Це підкреслює різницю між наявністю шифрування та наявністю довіри.

```
echo | openssl s_client -connect localhost:443 -servername
localhost
```

У виводі слід знайти: код верифікації, використаний шифр (Cipher), інформацію про власника сертифіката (subject).

3.2.4. Порівняння з довіреним сертифікатом. Для контрасту студент виконує ту саму команду для сайту з довіреним сертифікатом (наприклад, google.com). У цьому випадку *Verify return code* буде 0 (ok), що означає успішну верифікацію ланцюга довіри через кореневі CA, встановлені в системі. Це демонструє, чому у виробничих середовищах використовуються сертифікати від довірених центрів сертифікації (CA), а не самопідписані.

```
echo | openssl s_client -connect google.com:443 -servername
google.com
```

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов'язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім'я та по батькові студента, назва академічної групи, а також прізвище, ім'я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

Лабораторна робота № 10

Тема: Аналіз інцидентів інформаційної безпеки в Linux

Мета роботи: Набути практичних навичок аналізу журналів подій Linux з метою виявлення та первинної оцінки інцидентів інформаційної безпеки.

1 КЛЮЧОВІ ПОЛОЖЕННЯ

1.1 Моніторинг подій безпеки в операційних системах та архітектура системи журналювання в Linux

Сучасні інформаційні системи постійно піддаються різноманітним загрозам безпеки, що робить моніторинг подій та своєчасне виявлення інцидентів критично важливими завданнями для забезпечення цілісності, доступності та конфіденційності даних. У контексті операційної системи Linux, яка широко використовується як на серверах, так і в корпоративному середовищі, система журналювання виступає основним механізмом фіксації всіх значущих подій, пов'язаних із роботою системи, діями користувачів та спробами несанкціонованого доступу. Саме аналіз цих журналів дозволяє фахівцям з кібербезпеки своєчасно виявляти аномалії, досліджувати інциденти та формувати цифрові докази для подальшого розслідування. Без належного моніторингу навіть найсучасніші засоби захисту можуть виявитися неефективними, оскільки організація не зможе адекватно реагувати на загрози в режимі реального часу.

Моніторинг подій безпеки в Linux-системах базується на комплексному підході, що поєднує аналіз системних журналів, моніторинг мережевої активності та кореляцію різнорідних джерел інформації. Основою цього підходу є розуміння архітектури системи журналювання, яка в сучасних дистрибутивах Linux представлена двома ключовими компонентами: традиційною службою rsyslog та уніфікованою системою systemd-journald. Ці компоненти працюють паралельно або інтегровано, забезпечуючи різні рівні деталізації та гнучкості при збиранні інформації про події. Розуміння принципів роботи цих систем, форматів журналів та методів їх аналізу є фундаментальною компетенцією для будь-якого фахівця з безпеки інформаційних систем. Крім того, важливим аспектом є вміння інтегрувати дані з різних джерел для отримання цілісної картини інциденту, що дозволяє не лише виявити факт атаки, але й реконструювати її повний ланцюжок.

Система журналювання в операційних системах сімейства Linux еволюціонувала від простих текстових файлів до складних багаторівневих систем зберігання та обробки подій. Традиційно в Unix-подібних системах використовувалася служба syslog, яка забезпечувала централізоване збирання повідомлень від ядра системи, демонів та додатків у текстові файли, розташовані в каталозі */var/log/*. У сучасних дистрибутивах Linux, зокрема в Debian, Ubuntu та їх похідних, функціонал *syslog* реалізується через службу

rsyslog, яка є розширеною та більш гнучкою версією оригінального протоколу. Служба rsyslog підтримує фільтрацію повідомлень за пріоритетом та типом, їх маршрутизацію до різних файлів або віддалених серверів, а також забезпечує можливість використання шаблонів для форматування виводу. Одночасно з *rsyslog* у системах, що використовують systemd як систему ініціалізації, функціонує служба *systemd-journald*, яка зберігає журнали у бінарному форматі та надає потужний інтерфейс для їх запиту через утиліту *journalctl*.

Ключовою особливістю архітектури журналювання є ієрархічна структура каталогу */var/log/,* де кожна служба або підсистема може мати власний файл або підкаталог для зберігання подій. Серед найважливіших файлів журналів слід виділити *auth.log*, який містить усі події, пов'язані з автентифікацією та авторизацією користувачів, включаючи спроби входу через SSH, sudo та локальні сесії. Файл *syslog* містить загальні системні повідомлення від різноманітних демонів та служб, тоді як *kern.log* фіксує повідомлення безпосередньо від ядра операційної системи, що може включати інформацію про апаратні помилки, завантаження модулів та критичні події безпеки. Окремі служби, такі як веб-сервер Apache, система захисту *Fail2ban* або менеджер пакетів *dpkg*, створюють власні журнали у відповідних підкаталогах або файлах. Розуміння призначення кожного з цих файлів дозволяє аналітику безпеки ефективно локалізувати джерело проблеми та зібрати релевантну інформацію про інцидент. При цьому важливо враховувати, що журнали можуть ротуватися за допомогою утиліти *logrotate*, що призводить до створення архівованих версій файлів з відповідними суфіксами (.1, .2.gz тощо).

1.2 Формати журналів та структура записів подій

Записи в системних журналах Linux мають стандартизований формат, що полегшує їх автоматичний аналіз та обробку засобами скриптів або спеціалізованих систем управління подіями безпеки (SIEM). Типовий запис у текстовому журналі rsyslog складається з кількох обов'язкових полів: часова мітка події у форматі Mmm dd hh:mm:ss, де Mmm це скорочена назва місяця англійською мовою, dd це день місяця, а hh:mm:ss це час у форматі годин, хвилин та секунд; ім'я хоста, на якому сталася подія; ідентифікатор процесу або служби, що згенерувала повідомлення, часто у форматі службаname[PID]; та власне текст повідомлення, який може містити структуровану або неструктуровану інформацію залежно від служби. Наприклад, запис про невдалу спробу входу через SSH може виглядати як "Jan 25 14:32:15 server sshd[12345]: Failed password for invalid user admin from 192.168.1.100 port 54321 ssh2". У цьому прикладі чітко видно всі ключові елементи: часову мітку, ідентифікатор служби sshd, факт помилки автентифікації, ім'я користувача, IP-адресу джерела атаки та використаний порт.

Журнали *systemd-journald*, на відміну від текстових файлів *rsyslog*, зберігаються у бінарному форматі, що забезпечує більшу ефективність зберігання та швидкість пошуку, але вимагає використання спеціалізованої утиліти journalctl для їх перегляду. Записи у *journald* містять не лише

традиційні поля, такі як часова мітка та повідомлення, але й розширені метадані, включаючи ідентифікатор юніта systemd, пріоритет повідомлення за стандартом *syslog* (від 0-emergency до 7-debug), ідентифікатори процесу та потоку, а також контекстну інформацію про стан системи на момент події. Це дозволяє здійснювати більш точну фільтрацію та кореляцію подій, наприклад, переглядати всі повідомлення від конкретної служби за останню годину або знайти всі критичні помилки у визначений часовий проміжок. Важливою перевагою journald є також можливість зберігання структурованих даних у форматі ключ-значення, що спрощує автоматичний парсинг та інтеграцію з системами аналітики. Розуміння різниці між цими двома підходами до журналювання є критично важливим для ефективного моніторингу безпеки в сучасних Linux-системах.

1.3 Методи аналізу подій автентифікації

Автентифікація користувачів є одним із найбільш критичних аспектів безпеки будь-якої операційної системи, оскільки саме цей процес визначає, хто отримає доступ до системних ресурсів та даних. У Linux-системах всі спроби автентифікації, незалежно від їх результату та методу (локальний вхід, SSH, *sudo, su*), фіксуються у файлі *auth.log* або в журналах *journald* з відповідними мітками. Аналіз цих подій дозволяє виявити широкий спектр атак та підозрілої активності, від простих спроб підбору пароля до складніших сценаріїв компрометації облікових записів. Типовими індикаторами атаки є множинні невдалі спроби входу з однієї IP-адреси за короткий проміжок часу, спроби автентифікації під неіснуючими іменами користувачів, входи в нетиповий для легітимного користувача час доби, або автентифікація з географічно віддалених локацій, які не відповідають звичайним патернам роботи. Особливу увагу слід приділяти спробам входу під обліковим записом root, оскільки компрометація цього облікового запису надає зловмиснику повний контроль над системою.

Для ефективного аналізу подій автентифікації використовуються різноманітні інструменти командного рядка, що дозволяють швидко фільтрувати та агрегувати великі обсяги журнальних даних. Утиліта *grep* є базовим інструментом для пошуку конкретних патернів у текстових файлах, наприклад, команда grep “Failed password” */var/log/auth.log* дозволяє виділити всі записи про невдалі спроби введення пароля. Більш складні запити можна створювати за допомогою регулярних виразів, наприклад, для пошуку всіх спроб входу з конкретної підмережі або під певним набором імен користувачів. Утиліта *journalctl* надає ще більш потужні можливості фільтрації завдяки підтримці структурованих запитів за різними полями журналу: параметр *-u* дозволяє фільтрувати події за юнітом systemd (наприклад, *journalctl -u ssh* для відображення лише подій служби SSH), параметри –since та –until дозволяють обмежити часовий діапазон аналізу, а параметр *-p* дозволяє фільтрувати за рівнем пріоритету повідомлення. Комбінування цих підходів дозволяє аналітику безпеки швидко локалізувати релевантні події та виявити патерни,

характерні для конкретних типів атак. У таблиці 10.1 наведено основні команди для аналізу подій автентифікації та їх призначення.

Таблиця 10.1 - Команди для аналізу подій автентифікації в Linux

Команда	Призначення
grep "Failed password" /var/log/auth.log	Пошук усіх невдалих спроб автентифікації за паролем
grep "Accepted password" /var/log/auth.log	Пошук успішних автентифікацій користувачів
journalctl -u ssh	Перегляд усіх подій служби SSH через journald
journalctl -u ssh --since "1 hour ago"	Перегляд подій SSH за останню годину
journalctl --since "2025-01-25 10:00" --until "2025-01-25 12:00"	Перегляд подій у конкретному часовому діапазоні
grep "Invalid user" /var/log/auth.log	Виявлення спроб входу під неіснуючими іменами користувачів
last -f /var/log/wtmp	Перегляд історії успішних входів користувачів
lastb -f /var/log/btmp	Перегляд історії невдалих спроб входу
journalctl -p err	Перегляд усіх повідомлень рівня помилки та вище
grep "session opened" /var/log/auth.log	Виявлення фактів відкриття користувацьких сесій

1.4 Виявлення та аналіз атак типу brute-force

Атаки типу brute-force, або атаки грубої сили, являють собою один із найпоширеніших методів компрометації систем, доступних через мережу. Суть цього методу полягає у систематичному переборі можливих комбінацій облікових даних до тих пір, поки не буде знайдено правильну пару логіна та пароля. У контексті Linux-серверів найчастішою метою таких атак є служба SSH, яка типово працює на порту 22 та забезпечує віддалений доступ до системи. Автоматизовані інструменти зловмисників здатні генерувати сотні або навіть тисячі спроб автентифікації за хвилину, перебираючи популярні імена користувачів (*root, admin, user, test*) та поширені паролі з заздалегідь підготовлених словників. Характерною ознакою такої атаки у журналах є послідовність записів про невдалі спроби входу, що йдуть один за одним з мінімальними часовими інтервалами, часто з однієї або кількох IP-адрес, які можуть належати до ботнету. Іноді зловмисники використовують розподілені атаки, коли спроби входу надходять з великої кількості різних IP-адрес для уникнення автоматичного блокування за підозрілою активністю.

Виявлення *brute-force* атак вимагає систематичного моніторингу журналів автентифікації та розуміння нормальних патернів використання системи. Простий підхід полягає у підрахунку кількості невдалих спроб входу з конкретної IP-адреси за визначений проміжок часу, що можна здійснити за допомогою комбінації команд *grep, awk* та *sort*. Наприклад, команда *grep "Failed password" /var/log/auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -nr* дозволяє отримати список IP-адрес, відсортований за кількістю невдалих спроб

автентифікації. Якщо певна адреса фігурує у сотнях записів за короткий час, це є явною ознакою автоматизованої атаки. Більш просунутий аналіз може включати перевірку географічного розташування IP-адрес, що дозволяє виявити атаки з регіонів, де легітимні користувачі системи зазвичай не працюють. Для автоматизації захисту від таких атак широко використовується інструмент Fail2ban, який моніторить журнали в реальному часі та автоматично додає правила до брандмауера для блокування IP-адрес, що демонструють підозрілу активність. Розуміння механізмів виявлення та протидії brute-force атакам є фундаментальною навичкою для адміністраторів безпеки, оскільки такі атаки залишаються актуальною загрозою для будь-яких систем, доступних через мережу.

1.5 Кореляція подій із мережевою активністю

Аналіз лише системних журналів, хоча і є важливим компонентом розслідування інцидентів, не завжди надає повну картину того, що саме відбувалося в системі під час атаки. Для глибшого розуміння необхідно корелювати події безпеки з фактичною мережевою активністю, що дозволяє підтвердити або спростувати гіпотези, сформовані на основі журналів, а також виявити додаткові аспекти атаки, які могли не бути зафіксовані засобами журналювання. Мережевий трафік містить інформацію про фактичні з’єднання між системами, обсяги переданих даних, використані протоколи та патерни комунікації, що можуть вказувати на сканування портів, експлуатацію вразливостей або ексфільтрацію даних. Кореляція цих даних з подіями автентифікації дозволяє встановити прямий зв’язок між мережевою активністю та спробами компрометації системи, що є критично важливим для формування цифрових доказів та розуміння повного ланцюжка атаки.

Для перехоплення та аналізу мережевого трафіку в Linux-системах використовуються різноманітні інструменти, серед яких найпоширенішими є tcpdump та Wireshark. Утиліта *tcpdump* є консольним інструментом, який дозволяє перехоплювати пакети в реальному часі та зберігати їх у файли формату PCAP (Packet Capture) для подальшого аналізу. Основною перевагою tcpdump є його простота використання та мінімальні вимоги до ресурсів системи, що робить його ідеальним для швидкого захоплення трафіку на віддалених серверах або в умовах обмежених ресурсів. Команда tcpdump підтримує широкий набір фільтрів, що дозволяють захоплювати лише релевантний трафік, наприклад, пакети на конкретному порту, від певної IP-адреси або з певними прапорцями TCP. Wireshark, навпаки, є графічним інструментом, який надає потужні можливості візуального аналізу захоплених пакетів, включаючи декодування протоколів різних рівнів, реконструкцію TCP-сесій, статистичний аналіз трафіку та виявлення аномалій. Комбінація цих інструментів дозволяє аналітику безпеки як швидко захопити трафік у критичній ситуації за допомогою tcpdump, так і провести глибокий детальний аналіз у Wireshark на робочій станції.

Типовий сценарій кореляції подій включає наступні кроки: спочатку в журналах автентифікації виявляються підозрілі події, наприклад, серія невдалих спроб входу через SSH з певної IP-адреси; далі за допомогою tcpdump захоплюється мережевий трафік на порту 22 (SSH) за той самий часовий період; потім захоплений трафік аналізується у Wireshark для підтвердження фактів встановлення TCP-з'єднань з цієї IP-адреси та перевірки наявності аномальних патернів, таких як велика кількість коротких з'єднань або спроби сканування портів. На рівні мережевих пакетів можна побачити трьохстороннє рукостискання TCP (SYN, SYN-ACK, ACK), яке передує кожній спробі SSH-автентифікації, а також можливі аномалії, такі як незавершені з'єднання або великі затримки між пакетами. Аналіз розміру та послідовності пакетів може також вказувати на використання автоматизованих інструментів атаки, оскільки такі інструменти часто генерують характерні патерни трафіку, відмінні від ручної автентифікації. У таблиці 10.2 наведено основні команди tcpdump для захоплення та фільтрації мережевого трафіку.

Таблиця 10.2 - Команди tcpdump для моніторингу мережевої активності

Команда	Призначення
tcpdump -i any port 22	Перехоплення всього трафіку на порту 22 (SSH) на всіх мережевих інтерфейсах
tcpdump -i eth0 host 192.168.1.100	Перехоплення трафіку від/до конкретної IP-адреси на інтерфейсі eth0
tcpdump -i any port 22 -w capture.pcap	Збереження захопленого SSH-трафіку у файл формату PCAP
tcpdump -i any 'tcp[tcpflags] & (tcp-syn) != 0'	Захоплення лише TCP-пакетів з встановленим прапорцем SYN
tcpdump -i any src 192.168.1.100 and dst port 22	Захоплення трафіку з конкретної IP до порту 22
tcpdump -i any -n -c 100	Захоплення перших 100 пакетів без перетворення адрес у імена
tcpdump -i any port 22 -A	Відображення ASCII-вмісту пакетів на порту 22
tcpdump -r capture.pcap 'tcp.port == 22'	Читання збереженого файлу з фільтрацією за портом
tcpdump -i any net 192.168.1.0/24	Захоплення трафіку з/до всієї підмережі
tcpdump -i any -s 0 -w full.pcap	Захоплення повних пакетів без обмеження розміру

1.6 Формування цифрових доказів та реагування на інциденти

Виявлення інциденту безпеки є лише першим кроком у процесі реагування, за яким має слідувати збирання та збереження цифрових доказів, які можуть бути використані для подальшого розслідування, судового переслідування зловмисників або внутрішнього аудиту. Цифрові докази мають бути зібрані таким чином, щоб забезпечити їх цілісність, достовірність та прийнятність у якості доказової бази, що вимагає дотримання певних процедур та стандартів. Ключовими принципами збирання цифрових доказів є мінімізація впливу на оригінальні дані, документування всіх дій аналітика, забезпечення ланцюжка зберігання доказів та використання криптографічних

контрольних сум для підтвердження незмінності даних. У контексті аналізу інцидентів у Linux-системах цифрові докази можуть включати копії релевантних фрагментів журналів, PCAP-файли з мережевим трафіком, знімки стану системи на момент інциденту, а також метадані про файли, процеси та мережеві з'єднання.

Процес формування цифрових доказів починається з ідентифікації релевантних джерел інформації на основі попереднього аналізу інциденту. Якщо виявлено атаку через SSH, необхідно зберегти відповідні фрагменти файлу *auth.log*, вивід команд *journalctl* з фільтрацією за службою SSH та часовим діапазоном інциденту, а також PCAP-файл з мережевим трафіком на порту 22 за відповідний період. Для забезпечення цілісності цих файлів рекомендується обчислити їх криптографічні хеші за допомогою алгоритмів SHA-256 або SHA-512 одразу після створення копії, що дозволить у майбутньому підтвердити, що файли не були змінені. Команди для цього можуть включати sha256sum або md5sum, результати яких слід зберегти в окремому файлі разом з детальною інформацією про час, місце та обставини збирання доказів. Крім того, важливо зафіксувати контекст інциденту, включаючи опис системи, її конфігурацію, активні служби та мережеві налаштування на момент події, що може бути здійснено за допомогою команд на кшталт *uname -a, ip addr show, systemctl list-units* та інших.

Після збирання доказів наступним етапом є аналіз та формулювання висновків щодо характеру інциденту, його масштабу та потенційних наслідків. Цей процес вимагає корелювання даних з різних джерел для побудови повної хронології подій та визначення методів, використаних зловмисником. Наприклад, якщо в auth.log виявлено 500 невдалих спроб входу під іменем root з IP-адреси 203.0.113.50 протягом п'яти хвилин, а аналіз PCAP-файлу підтверджує наявність відповідних TCP-з'єднань з цієї адреси, можна зробити обґрунтований висновок про автоматизовану brute-force атаку. Важливо також оцінити, чи досягла атака своєї мети: чи були успішні спроби входу після невдалих, чи відбувалися підозрілі дії після компрометації облікового запису, чи було передано чутливі дані з системи. На основі цього аналізу формуються рекомендації щодо усунення наслідків інциденту та запобігання подібним атакам у майбутньому. Такі рекомендації можуть включати зміну паролів скомпрометованих облікових записів, блокування IP-адрес зловмисників у брандмауері, посилення політик автентифікації (наприклад, вимога використання ключів SSH замість паролів), впровадження системи автоматичного блокування після декількох невдалих спроб входу та регулярний моніторинг журналів на предмет подібної активності.

1.7 Інструменти автоматизації моніторингу та реагування

Ручний аналіз журналів є важливою навичкою, але в умовах великих обсягів даних та необхідності реагування в реальному часі стає неефективним. Для автоматизації процесів моніторингу безпеки та реагування на інциденти в Linux-системах використовуються спеціалізовані інструменти, які здатні

відстежувати журнали, виявляти підозрілі патерни та автоматично вживати заходів для блокування атак. Одним із найпопулярніших таких інструментів є Fail2ban, який працює як демон, що постійно моніторить вказані файли журналів на предмет певних патернів, що свідчать про зловмисну активність. Коли Fail2ban виявляє, що з певної IP-адреси надійшло занадто багато невдалих спроб автентифікації за короткий час, він автоматично додає правило до брандмауера (iptables або nftables), яке блокує весь трафік з цієї адреси на визначений період часу. Це дозволяє ефективно протидіяти brute-force атакам без необхідності постійного ручного втручання адміністратора.

Конфігурація Fail2ban базується на системі фільтрів та дій, де фільтри визначають регулярні вирази для пошуку підозрілих патернів у журналах, а дії описують, що саме робити при виявленні таких патернів. Наприклад, стандартний фільтр для SSH аналізує файл auth.log на наявність рядків, що містять “Failed password” або “Invalid user”, і якщо за визначений проміжок часу (зазвичай 10 хвилин) з однієї IP-адреси надходить більше певної кількості таких спроб (зазвичай 5), IP блокується на встановлений час (зазвичай від 10 хвилин до декількох годин). Конфігураційні файли Fail2ban зберігаються у каталозі /etc/fail2ban/ та включають основний файл jail.conf, який містить глобальні налаштування, та окремі файли фільтрів у підкаталозі filter.d/. Адміністратори можуть налаштовувати порогові значення, тривалість блокування та список служб, що моніторяться, відповідно до політик безпеки організації та специфіки використання системи.

Іншим важливим напрямком автоматизації є використання систем централізованого управління журналами та SIEM-систем (Security Information and Event Management), які дозволяють збирати журнали з множини систем, корелювати події між різними джерелами та виявляти складні багатоетапні атаки. Популярними рішеннями у цій категорії є ELK Stack (Elasticsearch, Logstash, Kibana), який забезпечує збирання, індексування та візуалізацію журнальних даних, а також Splunk, Graylog та інші комерційні та відкриті платформи. Ці системи дозволяють створювати складні правила кореляції, наприклад, виявляти ситуації, коли невдалі спроби входу через SSH з певної IP-адреси супроводжуються скануванням портів або спробами експлуатації вразливостей веб-сервера. Для навчальних цілей та початкового ознайомлення з принципами автоматизованого моніторингу достатньо розуміти базові можливості Fail2ban та вміти інтерпретувати його журнали, які зберігаються у файлі /var/log/fail2ban.log і містять інформацію про всі заблоковані IP-адреси та причини блокування.

1.8 Протоколи та технології віддаленого доступу як об’єкти атак

Служби віддаленого доступу, зокрема SSH (Secure Shell), є критично важливими компонентами сучасних IT-інфраструктур, оскільки вони забезпечують можливість адміністрування серверів та робочих станцій без фізичної присутності біля обладнання. Водночас саме ці служби є одними з найпривабливіших цілей для зловмисників, оскільки успішна компрометація

облікового запису з правами віддаленого доступу надає атакуючому широкі можливості для подальших дій у системі. Протокол SSH працює на порту 22 за замовчуванням та забезпечує шифроване з'єднання між клієнтом та сервером, що захищає від перехоплення облікових даних під час передачі по мережі. Однак сам факт шифрування не захищає від атак підбору пароля, якщо система налаштована на прийняття парольної автентифікації замість більш безпечної автентифікації за допомогою криптографічних ключів.

Типова SSH-сесія починається з встановлення TCP-з'єднання між клієнтом та сервером через трьохстороннє рукостискання, після чого відбувається обмін версіями протоколу SSH та узгодження криптографічних параметрів сесії. На цьому етапі сервер надсилає клієнту свій публічний ключ, який клієнт може перевірити для запобігання атакам типу man-in-the-middle. Далі відбувається процес автентифікації, під час якого клієнт надає облікові дані у формі пароля або приватного ключа, що відповідає публічному ключу, збереженому на сервері. Усі ці етапи фіксуються у системних журналах з різним рівнем деталізації, що дозволяє аналітику безпеки реконструювати послідовність подій навіть після завершення сесії. Розуміння структури SSH-з'єднання та типових патернів легітимного використання є необхідним для відрізнення нормальної активності від атак, оскільки деякі законні сценарії (наприклад, автоматизовані скрипти моніторингу) також можуть генерувати велику кількість з'єднань.

Захист SSH-служби від атак включає декілька рівнів заходів безпеки, які можуть бути застосовані одночасно для створення ешелонованої оборони. На рівні конфігурації служби рекомендується вимкнути автентифікацію для користувача root (параметр PermitRootLogin no у файлі */etc/ssh/sshd_config*), що змушує зловмисників спочатку компрометувати звичайний обліковий запис, а потім здійснювати підвищення привілеїв. Використання автентифікації на основі публічних ключів замість паролів (PasswordAuthentication no) практично повністю усуває ризик успішної brute-force атаки, оскільки підбір криптографічного ключа є обчислювально неможливим за розумний час. Зміна стандартного порту SSH з 22 на нестандартний значно зменшує кількість автоматизованих атак, оскільки більшість ботів сканують лише стандартні порти, хоча це є прикладом безпеки через невідомість і не повинно розглядатися як основний засіб захисту. На рівні мережі можна використовувати обмеження доступу за IP-адресами через брандмауер, дозволяючи підключення лише з відомих та довірених адрес, або впроваджувати VPN для створення додаткового рівня автентифікації перед доступом до SSH-служби.

1.9 Практичні аспекти використання Wireshark для аналізу безпеки

Wireshark є одним із найпотужніших інструментів для аналізу мережевого трафіку, який надає детальний погляд на всі рівні мережевого стеку від фізичного до прикладного. На відміну від tcpdump, який орієнтований на швидке захоплення пакетів у консольному режимі, Wireshark пропонує

графічний інтерфейс з можливостями кольорового кодування пакетів, декодування протоколів, статистичного аналізу та візуалізації потоків даних. При відкритті PCAP-файла у Wireshark користувач бачить список усіх захоплених пакетів з основними параметрами: номер пакета, часова мітка, IP-адреси джерела та призначення, протокол та коротка інформація про вміст. Кожен пакет може бути розгорнутий для перегляду його повної структури, включаючи заголовки всіх рівнів мережевого стеку та корисне навантаження у шістнадцятковому та ASCII-форматах.

Ключовою функцією Wireshark для аналізу інцидентів безпеки є система фільтрів відображення, яка дозволяє швидко ізолювати релевантний трафік серед тисяч або мільйонів пакетів. Фільтри можуть бути застосовані за протоколом (наприклад, ssh для відображення лише SSH-трафіку), за IP-адресами (ip.src == 192.168.1.100 для пакетів з конкретного джерела), за портами (tcp.port == 22), або за комбінацією умов з використанням логічних операторів. Для аналізу SSH-атак особливо корисними є фільтри, що виділяють пакети встановлення з'єднання (tcp.flags.syn == 1) або пакети з певними розмірами, що можуть вказувати на передачу облікових даних. Wireshark також підтримує функцію Follow TCP Stream, яка дозволяє реконструювати повний обмін даними між клієнтом та сервером у межах одного TCP-з'єднання, що надзвичайно корисно для розуміння послідовності подій під час атаки або легітимної сесії.

Статистичні можливості Wireshark дозволяють виявляти аномалії та патерни, які можуть бути непомітними при перегляді окремих пакетів. Меню Statistics надає доступ до різноманітних звітів, включаючи графік інтенсивності трафіку у часі (IO Graph), розподіл протоколів (Protocol Hierarchy), статистику бесід між хостами (Conversations) та інформацію про endpoints. Для виявлення brute-force атак особливо корисним є звіт Conversations, який показує кількість пакетів та обсяг даних, передані між кожною парою IP-адрес, що дозволяє швидко ідентифікувати джерела аномально високої активності. Графік IO Graph може показати раптові сплески трафіку на порту SSH, що корелюють з часом атаки, зафіксованим у системних журналах. У таблиці 10.3 наведено основні фільтри Wireshark, корисні для аналізу SSH-трафіку та інцидентів безпеки.

Таблиця 10.3 - Фільтри Wireshark для аналізу SSH-трафіку та подій безпеки

Фільтр	Призначення
ssh	Відображення всіх пакетів протоколу SSH
tcp.port == 22	Відображення всього TCP-трафіку на порту 22
ip.src == 192.168.1.100	Пакети від конкретної IP-адреси джерела
ip.dst == 192.168.1.1	Пакети до конкретної IP-адреси призначення
tcp.flags.syn == 1 && tcp.flags.ack == 0	Початкові пакети встановлення з'єднання (SYN)
tcp.flags.reset == 1	Пакети скидання з'єднання (RST)
tcp.analysis.retransmission	Виявлення повторних передач (можливі проблеми мережі)
frame.time >= "2025-01-25 10:00:00"	Фільтрація за часовою міткою

ssh && ip.src == 203.0.113.50	Комбінований фільтр: SSH-трафік з певної IP
tcp.stream eq 5	Перегляд конкретного TCP-потоку за номером
tcp.connection.syn	Нові TCP-з’єднання
!(arp or icmp or dns)	Виключення допоміжних протоколів з відображення

1.10 Методологія реагування на інциденти інформаційної безпеки

Ефективне реагування на інциденти інформаційної безпеки вимагає структурованого підходу, який забезпечує мінімізацію збитків, збереження доказів та швидке відновлення нормальної роботи системи. Міжнародні стандарти, такі як NIST SP 800-61 (Computer Security Incident Handling Guide), визначають кілька ключових етапів процесу реагування: підготовка, виявлення та аналіз, стримування, ліквідація та відновлення, а також аналіз після інциденту. На етапі підготовки організація розробляє політики безпеки, впроваджує засоби моніторингу, навчає персонал та створює процедури реагування. Виявлення та аналіз включають моніторинг подій безпеки, їх класифікацію за рівнем критичності та глибоке розслідування для визначення масштабу та природи інциденту. Стримування має на меті ізолювати скомпрометовані системи для запобігання поширенню атаки, тоді як ліквідація включає видалення зловмисного програмного забезпечення, закриття вразливостей та відновлення систем до безпечного стану.

У контексті Linux-систем процес реагування на інцидент, виявлений через аналіз журналів автентифікації та мережевого трафіку, може включати декілька конкретних технічних дій. Якщо виявлено активну brute-force атаку з певної IP-адреси, першим кроком стримування є блокування цієї адреси за допомогою брандмауера командою на кшталт iptables -A INPUT -s 203.0.113.50 -j DROP, що негайно припиняє всі спроби з’єднання з цього джерела. Паралельно необхідно перевірити, чи не вдалося зловмиснику здійснити успішний вхід, проаналізувавши журнали на наявність записів “Accepted password” або “session opened” від відповідної IP-адреси. Якщо компрометація підтверджена, потрібно негайно змінити паролі всіх потенційно скомпрометованих облікових записів, перевірити наявність підозрілих процесів за допомогою команд ps, top або htop, проаналізувати активні мережеві з’єднання командою netstat або ss, та перевірити зміни у конфігураційних файлах і наявність нових облікових записів користувачів.

Документування всіх дій під час реагування на інцидент є критично важливим як для подальшого аналізу, так і для можливого судового переслідування. Кожна команда, виконана адміністратором під час розслідування, повинна бути зафіксована разом з часовою міткою та отриманими результатами, що дозволяє відтворити процес розслідування та підтвердити коректність зроблених висновків. Для цього можна використовувати команду script, яка зберігає повну копію сесії терміналу у файл, або систематично зберігати виводи важливих команд у текстові файли за допомогою перенаправлення виводу. Після успішного стримування та ліквідації інциденту проводиться ретроспективний аналіз для визначення

першопричин компрометації, оцінки ефективності процесу реагування та формулювання рекомендацій щодо запобігання подібним інцидентам у майбутньому. Цей аналіз може призвести до оновлення політик безпеки, впровадження додаткових технічних засобів захисту або перегляду процедур управління доступом.

2 КЛЮЧОВІ ПИТАННЯ

1. Які основні компоненти системи журналювання використовуються в сучасних Linux-дистрибутивах та в чому полягають їх функціональні відмінності? Поясніть різницю між rsyslog та systemd-journald з точки зору формату зберігання даних та методів доступу до журналів.
2. Які файли журналів у каталозі /var/log/ містять інформацію про події автентифікації, і яку конкретну інформацію можна отримати з кожного з них? Опишіть структуру типового запису у файлі auth.log та поясніть призначення кожного поля.
3. Які характерні ознаки автоматизованої brute-force атаки на SSH-службу можна виявити при аналізі системних журналів? Як відрізнити автоматизовану атаку від помилок легітимного користувача, який забув пароль?
4. Які команди та утиліти використовуються для фільтрації та аналізу подій у журналах Linux, і як за їх допомогою знайти всі невдалі спроби SSH-автентифікації за останню годину? Наведіть приклади використання grep та journalctl для різних сценаріїв аналізу.
5. Яке призначення утиліти tcpdump та як її використовувати для перехоплення SSH-трафіку з збереженням результатів у файл формату PCAP? Поясніть синтаксис базової команди tcpdump для моніторингу конкретного порту та IP-адреси.
6. Як здійснюється кореляція подій із системних журналів з даними мережевого трафіку для підтвердження факту атаки? Опишіть процес зіставлення записів у auth.log з TCP-з’єднаннями, виявленими у PCAP-файлі.
7. Які можливості надає Wireshark для візуального аналізу захопленого мережевого трафіку, і які фільтри доцільно використовувати при дослідженні SSH-інцидентів? Поясніть, як використовувати функцію Follow TCP Stream для реконструкції сесії.
8. Що таке цифрові докази в контексті розслідування інцидентів інформаційної безпеки, і які вимоги ставляться до процесу їх збирання та зберігання? Як забезпечити цілісність зібраних журналів та PCAP-файлів за допомогою криптографічних хеш-функцій?
9. Які технічні та організаційні заходи можуть бути вжиті для захисту SSH-служби від brute-force атак? Опишіть принцип роботи інструменту Fail2ban та його роль в автоматизованому реагуванні на підозрілу активність.
10. Які етапи включає методологія реагування на інциденти інформаційної безпеки згідно з міжнародними стандартами, і які конкретні дії мають бути виконані на кожному етапі при виявленні активної атаки на Linux-

систему? Чому важливе документування всіх дій під час розслідування інциденту?

3 ЛАБОРАТОРНЕ ЗАВДАННЯ

3.1 Ознайомлення з системою журналів подій Linux

Сучасні операційні системи Linux реалізують комплексну систему журналювання, яка фіксує події безпеки, стан служб, помилки ядра, спроби входу користувачів та інші важливі події. Централізоване зберігання цих даних у каталозі */var/log/* забезпечує можливість проведення аудиту, розслідування інцидентів та підвищення рівня захищеності системи. У Debian-подібних дистрибутивах основними компонентами журналювання є служби rsyslog (традиційна система *syslog*) та *systemd-journald* (уніфікована система журналів systemd). Для ефективного аналізу подій необхідно розуміти призначення ключових файлів журналів, таких як *auth.log* (події автентифікації), *syslog* (загальні системні події), *kern.log* (повідомлення ядра), а також вміти перевіряти стан служб журналювання. Ця частина лабораторної роботи спрямована на формування загального уявлення про структуру системних журналів та їх роль у забезпеченні безпеки.

3.1.1. Перегляд доступних журналів системи. Перед початком глибокого аналізу подій безпеки необхідно отримати повну картину доступних журналів у системі. Каталог */var/log/* містить як загальні системні файли журналів, так і спеціалізовані логи окремих служб (наприклад, *apache2/, fail2ban.log, dpkg.log*). Огляд цього каталогу дозволяє зрозуміти, які компоненти системи активно ведуть журналювання, де шукати потрібну інформацію та який обсяг даних уже зібрано. Особливу увагу слід приділити наявності ключових файлів, таких як *auth.log, syslog*, а також їх розміру та часу останнього оновлення — це дає уявлення про активність системи та потенційні джерела інцидентів.

```
ls -l /var/log/
```

Після виконання команди слід проаналізувати список файлів у каталозі */var/log/*. Зверніть увагу на наявність файлів *auth.log, syslog, kern.log* та, за наявності, *fail2ban.log*. Зафіксуйте розмір кожного з них та часову мітку останнього запису. Якщо файл *auth.log* відсутній або порожній, це може свідчити про те, що служба rsyslog неактивна або журналювання автентифікації вимкнено. У такому випадку рекомендується перевірити стан служби *rsyslog* за допомогою команди *systemctl status rsyslog*.

3.2 Аналіз подій автентифікації через системні журнали

Події автентифікації є одним із найважливіших джерел інформації при розслідуванні інцидентів безпеки. У Linux-системах всі спроби входу — як

успішні, так і невдалі — фіксуються у файлі */var/log/auth.log* (при використанні rsyslog) або в журналах *systemd-journald*. Ці дані дозволяють виявити автоматизовані атаки типу brute-force, спроби підбору облікових даних або несанкціонований доступ з віддалених IP-адрес. Для ефективного аналізу використовуються як класичні інструменти (*grep, tail*), так і сучасні утиліти, такі як *journalctl*, яка надає уніфікований інтерфейс для перегляду журналів systemd. У цій частині студент навчиться виявляти підозрілу активність, визначати джерела атак та оцінювати їх інтенсивність.

3.2.1. Аналіз журналів SSH-автентифікації. Служба SSH часто стає метою автоматизованих атак через її широке використання для віддаленого управління системами. Усі спроби підключення через SSH, незалежно від результату, реєструються у системних журналах. Команда *journalctl -u ssh* дозволяє відфільтрувати події, пов'язані саме зі службою SSH, а команда grep "Failed password" */var/log/auth.log* — виділити лише невдалі спроби входу. Це дає можливість швидко виявити підозрілу активність, зокрема множинні спроби входу з однієї IP-адреси або під різними іменами користувачів.

```
sudo journalctl –u ssh
sudo grep "Failed password" /var/log/auth.log
```

У виводі команд слід звернути увагу на такі параметри: IP-адреса джерела запиту, ім'я користувача, під яким здійснювалася спроба входу, та точна часова мітка події. Наявність сотень спроб входу за короткий проміжок часу, особливо під ім'ям *root*, є явною ознакою автоматизованої атаки. Зафіксуйте принаймні три приклади таких подій для подальшого аналізу.

3.2.2. Визначення джерела підозрілої активності за часовими параметрами. Для оперативного реагування на інцидент необхідно знати, коли саме відбулися підозрілі події. Інструмент *journalctl* надає можливість фільтрувати події за часовим діапазоном, що моделює ситуацію реального часу. Наприклад, команда *--since "10 minutes ago*" дозволяє переглянути всі події, зареєстровані за останні 10 хвилин. Це особливо корисно під час активної атаки, коли потрібно швидко зібрати докази та ухвалити рішення щодо блокування джерела.

```
sudo journalctl --since "10 minutes ago"
```

Проаналізуйте отриманий вивід на наявність записів, пов'язаних зі службою SSH. Якщо виявлено невдалі спроби входу, зафіксуйте IP-адресу джерела та кількість спроб. Порівняйте ці дані з результатами попереднього кроку. Якщо активність продовжується, це свідчить про тривалу атаку, що вимагає негайних заходів (наприклад, тимчасового блокування IP через *iptables* або *fail2ban*).

3.3 Кореляція подій із мережевою активністю. Журнали автентифікації надають інформацію про що сталося, але не завжди пояснюють як. Для повної реконструкції інциденту необхідно проаналізувати мережевий трафік, який супроводжував події безпеки. Це дозволяє підтвердити факт з'єднання з певної IP-адреси, виявити аномальні патерни (наприклад, SYN-флуд) або множинні паралельні з'єднання. У цій частині студент використовує інструменти *tcpdump* для перехоплення трафіку та Wireshark для його візуального аналізу, формуючи цифрові докази, які можуть бути використані в подальшому розслідуванні.

3.3.1. Перехоплення мережевого трафіку на порту SSH. Інструмент tcpdump дозволяє перехоплювати мережеві пакети в реальному часі. У цьому кроці студент запускає перехоплення трафіку на порту 22 (стандартний порт SSH) і одночасно імітує спробу входу (наприклад, через локальне підключення). Це дозволяє побачити, як виглядає TCP-з'єднання на рівні пакетів, і підтвердити кореляцію між подіями у *auth.log* та фактичною мережевою активністю.

```
# На сервері (в окремому терміналі):
sudo tcpdump –i any port 22
# На тому ж хості (імітація клієнта):
ssh fakeuser@localhost
```

У виводі tcpdump має з'явитися послідовність TCP-пакетів: SYN → SYN-ACK → ACK, що свідчить про встановлення з'єднання. Навіть якщо автентифікація завершилася невдало, сам факт з'єднання буде зафіксовано. Це підтверджує, що подія у *auth.log* має мережевий контекст, що є важливим елементом цифрових доказів.

3.3.2. Візуальний аналіз трафіку за допомогою Wireshark. Для глибшого аналізу та зберігання цифрових слідів студент зберігає перехоплений трафік у файл у форматі PCAP за допомогою tcpdump, а потім відкриває його у графічному аналізаторі Wireshark. Це дозволяє детально вивчити структуру пакетів, перевірити наявність аномалій (наприклад, великої кількості з'єднань за короткий час) та зробити скріншоти для включення до звіту.

```
# Запис трафіку у файл:
sudo tcpdump –i any port 22 –w ssh_traffic.pcap
```

Після завершення запису (натисніть Ctrl+C) відкрийте файл у Wireshark:

```
wireshark ssh_traffic.pcap
```

У Wireshark проаналізуйте наступне:

- наявність встановленого TCP-з'єднання (трьохстороннього рукостискання);
- наявність спроб передачі даних після встановлення з'єднання;

- наявність ознак автоматизованої атаки, зокрема декількох з’єднань з однієї IP-адреси протягом секунди.

3.4. Формування висновків щодо характеру інциденту. На завершальному етапі лабораторної роботи студент узагальнює всі зібрані дані та формує професійні висновки щодо характеру виявленого інциденту. Аналіз охоплює джерело атаки, її масштаб, часові параметри активності, а також потенційні наслідки успішного здійснення несанкціонованого доступу. На основі журналів *auth.log*, виводу команди *journalctl* та перехопленого мережевого трафіку у форматі PCAP-файлу студент визначає, чи мали місце ознаки автоматизованої атаки, наприклад множинні спроби входу за короткий проміжок часу. Також виконується ідентифікація джерела атаки — зокрема IP-адреси та імені користувача, під яким здійснювалися спроби автентифікації.

Мережевий трафік, зафіксований за допомогою *tcpdump* та проаналізований у Wireshark, слугує підтвердженням факту встановлення з’єднання з виявленою IP-адресою. Це дозволяє корелювати події у системних журналах із реальною мережевою активністю, що є важливим елементом формування достовірних цифрових доказів. У разі успішної атаки можливими наслідками могли б стати несанкціонований доступ до системи, витік конфіденційних даних, встановлення шкідливого програмного забезпечення або використання компрометованої системи як платформи для подальших атак.

Отримані навички мають значну практичну цінність: вміння аналізувати системні журнали, зіставляти події безпеки з мережевим трафіком та документувати цифрові сліди є ключовими компетентностями фахівця з кібербезпеки. Саме ці дії лежать в основі ефективного реагування на інциденти, проведення розслідувань компрометації та постійного підвищення рівня захищеності інформаційних систем.

4 ЗМІСТ ПРОТОКОЛУ

Протокол лабораторної роботи оформлюється відповідно до загальноприйнятих вимог до навчальної документації з використанням шрифту Times New Roman розміром 12 pt та міжрядкового інтервалу 1,5. Протокол має чітку структуру та складається з трьох обов’язкових частин, які подаються у визначеній послідовності.

Вступна частина містить основні відомості про виконану лабораторну роботу. У ній зазначаються тема та мета роботи, прізвище, ім’я та по батькові студента, назва академічної групи, а також прізвище, ім’я та по батькові викладача. Мета лабораторної роботи формулюється відповідно до навчальних цілей дисципліни та вимог методичних рекомендацій. У цій же частині наводяться відповіді на ключові питання, передбачені завданням до роботи. Відповіді мають бути чіткими, лаконічними та логічно обґрунтованими, базуватися на теоретичному матеріалі розділу «Ключові положення» і демонструвати розуміння студентом основних понять теми роботи.

Основна частина протоколу присвячена безпосередньому виконанню лабораторного завдання. Вона починається з короткого опису робочого середовища, у якому виконувалася робота: зазначається версія операційної системи, тип та конфігурація віртуальної машини, а також стан системи на момент початку виконання завдання (наявність необхідних пакетів, служб, мережевих налаштувань). Далі послідовно описується виконання кожного етапу лабораторної роботи відповідно до індивідуального варіанта студента. Для кожного кроку подається короткий опис виконуваних дій, наводяться використані команди командного рядка або фрагменти скриптів із поясненням їх призначення, синтаксису та умов застосування, а також фіксуються отримані результати. Результати виконання підтверджуються виводом командного рядка, змінами у файловій системі, станом процесів або служб і супроводжуються відповідними скріншотами.

У висновковій частині студент формулює узагальнені висновки щодо отриманих теоретичних знань, практичних умінь і професійних компетенцій, набутих у процесі виконання лабораторної роботи. Аналізуються типові помилки або труднощі, що виникали під час виконання завдання – зокрема, помилки в синтаксисі команд, неправильне налаштування прав доступу, некоректна конфігурація служб безпеки, проблеми з генерацією чи імпортом криптографічних ключів, а також описуються способи їх усунення.

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Горбенко В. І., Лісняк А. О. Безпека програм та даних : навчальний посібник для здобувачів ступеня вищої освіти бакалавра спеціальності 121 «Інженерія програмного забезпечення». — Запоріжжя : ЗНУ, 2022. — 72 с.

2. Сенів М. М., Яковина В. С. Безпека програм та даних : навчальний посібник. — Львів : Львівська політехніка, 2015. — 256 с.

3. Дем'яненко В. А., Кузнецова Ю. А. Безпека програм та даних [Електронний ресурс] : навчальний посібник до лабораторного практикуму. — Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «ХАІ», 2021. — 95 с.

4. Євсеєв С. П., Мілов О. В., Король О. Г. Лабораторний практикум з основ криптографічного захисту [Електронний ресурс] : навчальний посібник. — Харків : ХНЕУ ім. С. Кузнеця, 2020. — 222 с.

5. Технології захисту інформації в інформаційно-телекомунікаційних системах : навчальний посібник / А. В. Жилін, О. М. Шаповал, О. А. Успенський ; ІСЗЗІ КПІ ім. Ігоря Сікорського. — Київ : Вид-во «Політехніка» КПІ ім. Ігоря Сікорського, 2021. — 213 с.

6. Кузнецов О. О. Захист інформації в інформаційних системах : навчальний посібник. — Харків : ХНЕУ, 2018. — 510 с.

7. Мамченко С. М., Козюра В. Д., Бровко В. Д. Комплексні системи захисту інформації : навчальний посібник. — Київ : Національна академія СБУ, 2018. — 372 с.

8. Остапов С. Є., Євсеєв С. П., Король О. Г. Технології захисту інформації. — Чернівці : Видавничий дім «Родовід», 2014. — 428 с.

9. Згуровський М. Проблеми інформаційної безпеки в Україні, шляхи їх вирішення // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. — Київ, 2018. — С. 10–14.