НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.4:004.492

ПОГОДЖЕНО
Декан факультету
Інформаційних технологій
_____/ Болбот І.В., д.т.н, проф. /
підпис ПІБ, вчене звання і ступінь
«___»_____2024 р.

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.п.н., доцент. /
підпис ПІБ, вчене звання і ступінь
«___»_____2024 р.

МАГІСТЕРСЬКА РОБОТА

На тему: «Розробка та дослідження комп’ютерних систем для безпечного пентесту за допомогою Raspberry Pi»

Спеціальність: 123 «Комп’ютерна інженерія»

Освітня програма: Комп’ютерні системи захисту інформації

Керівник дипломного проекту: _____ / Лахно В.А. /
підпис ПІБ

Виконав: _____ / Панасенко С.А. /
підпис ПІБ

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

«ЗАТВЕРДЖУЮ»
завідувач кафедри
комп'ютерних систем, мереж та кібербезпеки
_____/ Касаткін Д.Ю., к.п.н., доцент. /
підпис ПІБ, вчене звання і ступінь
«__» _____ 2024 р.

ЗАВДАННЯ

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ РОБОТИ СТУДЕНТУ

Панасенку Станіславу Анатолійовичу
(прізвище, ім'я, по батькові)

Спеціальність (напрям підготовки): 123 «Комп'ютерна інженерія».
Освітня програма: комп'ютерні системи захисту інформаці

Тема магістерської роботи: «Розробка та дослідження комп'ютерних систем для безпечного пентесту за допомогою Raspberry Pi»
затверджена наказом ректора НУБІП України від " 1 " листопада 2023 № 1859 "С"
Термін подання завершеної роботи на кафедру
Вихідні дані до магістерської роботи: Raspberry Pi з встановленим програмним забезпеченням для сканування мереж (Nmap, OpenVAS), сервер для VPN тунелювання з використанням OpenVPN та налаштованим Telegram ботом для управління Raspberry Pi
Перелік питань, що підлягають дослідженню:

1. Аналіз предметної області для дослідження методів тестування на проникнення з використанням малопотужних пристроїв.
2. Проектування системи для безпечного пентесту з використанням Raspberry Pi та VPN тунелювання через Telegram бот.
3. Налаштування та тестування вразливої мережі з метою перевірки ефективності запропонованої системи безпечного пентесту.

Дата видачі завдання " 1 " листопада 2023 р.

Керівник магістерської роботи _____ / Лахно В.А. д.т.н., професор /
(підпис) (ПІБ, вчене звання і ступінь)

Завдання прийняв до виконання _____ / Панасенко С.А. /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Постановка задачі магістерської роботи		Виконано
2	Аналіз предметної області		Виконано
3	Проектування системи		Виконано
4	Реалізація системи		Виконано
5	Тестування розробленої системи		Виконано
6	Оформлення пояснювальної записки		Виконано
7	Оформлення графічного матеріалу		Виконано

Студент _____ / Панасенко С.А. /
підпис ПІБ

Керівник проекту (роботи) _____ / Лахно В.А. /
підпис ПІБ

РЕФЕРАТ

Пояснювальна записка: 122 с., 40 рис., 1 таблиці, 13 використаних джерела.

RASPBERRY PI, ПЕНТЕСТ, БЕЗПЕКА, СКАНУВАННЯ МЕРЕЖІ, TELEGRAM БОТ, OPENVPN, OPENVAS, NMAP, ВРАЗЛИВА МАШИНА, VPN ТУНЕЛЮВАННЯ

Мета роботи – створення системи для автоматизованого тестування на проникнення на основі Raspberry Pi, що включає віддалене управління через Telegram бот, забезпечення безпечного тунелювання через VPN та інтеграцію інструментів для сканування мереж, таких як Nmap і OpenVAS.

Об’єкт – система автоматизованого тестування на проникнення з використанням малопотужних пристроїв.

Предмет – програмні рішення з відкритим кодом для пентесту, сканування мереж та управління системами кібербезпеки.

Робота складається з трьох розділів.

У першому розділі проведено аналіз предметної області, розглянуто сучасні кіберзагрози, переваги використання Raspberry Pi у кібербезпеці, можливості Telegram-ботів для управління безпековими системами, а також виконано порівняння OpenVPN з іншими рішеннями для створення захищених з'єднань. Проведено аналіз аналогічних проєктів, що дозволило обґрунтувати вибір технологій для реалізації системи.

У другому розділі здійснено проектування та реалізацію системи для безпечного тестування на проникнення на основі Raspberry Pi. Описано архітектуру системи, налаштування центрального сервера з використанням OpenVPN для створення захищеного тунелю, встановлення Kali Linux на Raspberry Pi, вибір та інтеграцію інструментів для сканування, зокрема Nmap і OpenVAS, а також створення Telegram-бота для віддаленого управління. Для тестування системи налаштовано вразливу машину Metasploitable3.

У третьому розділі проведено тестування системи, аналіз результатів сканування мереж за допомогою Nmap та OpenVAS, а також оцінку

продуктивності Raspberry Pi в умовах підвищеного навантаження. Результати підтверджують ефективність системи, її придатність для виявлення вразливостей та автоматизації пентесту.

У результаті виконання магістерської роботи розроблено та протестовано систему для безпечного тестування на проникнення, яка може використовуватися в умовах обмежених ресурсів. Розроблено рекомендації щодо її впровадження та використання для підвищення рівня кібербезпеки.

ЗМІСТ

РЕФЕРАТ 4
ВСТУП 6
1 АНАЛІТИЧНИЙ ОГЛЯД 8
1.1 Огляд сучасних кіберзагроз та атак 8
1.2 Використання Raspberry Pi в кібербезпеці: переваги, обмеження та порівняння з альтернативами 26
1.3 Telegram-бот як засіб управління системами кібербезпеки 34
1.4 Використання OpenVPN: переваги, обмеження та порівняння з альтернативами 40
1.5 Порівняння з аналогічними проєктами 46
2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ БЕЗПЕЧНОГО ПЕНТЕСТУ З ВИКОРИСТАННЯМ RASPBERRY PI 50
2.1 Налаштування центрального сервера 50
2.1.1 Вибір хостингу та створення сервера на Kamatera 50
2.1.2 Підключення до сервера та оновлення системи 54
2.1.3 Встановлення необхідних пакетів 56
2.2 Налаштування OpenVPN на центральному сервері 57
2.2.1 Клонування репозиторію OpenVPN-інсталятора 57
2.2.2 Запуск скрипту встановлення OpenVPN 57
2.2.3 Налаштування конфігураційних файлів OpenVPN 59
2.2.4 Перезапуск служби OpenVPN 61
2.3 Налаштування та перевірка підключення Raspberry Pi до сервера 62
2.3.1 Встановлення Kali Linux на Raspberry Pi за допомогою Raspberry Pi Imager 62
2.3.2 Налаштування OpenVPN на Raspberry Pi 67
2.3.3 Налаштування автозапуску VPN-клієнта 68
2.3.4 Перевірка стану VPN-з'єднання 68

2.4 Встановлення та налаштування інструментів для сканування на Raspberry Pi 70
2.4.1 Налаштування OpenVAS 70
2.4.2 Налаштування автоматичного сканування мережі на Raspberry Pi 72
2.5 Розробка, налаштування та використання Telegram-бота 75
2.5.1 Створення Telegram-бота 75
2.5.2 Налаштування Python-середовища для Telegram-бота 75
2.5.3 Основні функції Telegram-бота 76
2.6 Налаштування вразливої машини для тестування внутрішньої мережі . 109
2.6.1 Завантаження та налаштування Metasploitable3 109
3 ПОБУДОВА МОДЕЛІ ЗАХИЩЕНОЇ МЕРЕЖІ В НАВЧАЛЬНОМУ ЗАКЛАДІ 112
3.1 Запуск сканування через Telegram-боту 112
3.2 Порівняння результатів сканування 113
3.3 Аналіз продуктивності та навантаження на систему 116
ВИСНОВКИ 119
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 120

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

VPN — Віртуальна приватна мережа (Virtual Private Network), технологія, що забезпечує створення захищених каналів зв’язку для передавання даних через загальнодоступні або приватні мережі.

Raspberry Pi — Одноплатний мікрокомп’ютер, що використовується для виконання різноманітних обчислювальних задач, зокрема у сфері кібербезпеки, мережевого моніторингу та тестування.

Nmap — Програмний інструмент для сканування мереж (Network Mapper), що дозволяє виявляти відкриті порти, запущені сервіси, їх версії та базові вразливості.

OpenVAS — Система для виявлення вразливостей (Open Vulnerability Assessment System), що забезпечує комплексний аналіз безпеки мереж та серверів.

Telegram бот — Програмний агент у месенджері Telegram, який автоматизує взаємодію з користувачем через команди та відповіді на запити.

OpenVPN — Програмне забезпечення для організації захищеного VPN-тунелювання, яке забезпечує шифрування даних і безпечний доступ до мережі.

VirtualBox — Система для віртуалізації, яка дозволяє створювати та запускати віртуальні машини на фізичному пристрої. Застосовується для тестування у відокремленому середовищі.

Metasploitable 3 — Вразлива віртуальна машина, що використовується для навчання кібербезпеці та тестування на проникнення.

Пентест — Тестування на проникнення, метод оцінки безпеки системи шляхом моделювання атак зловмисників для виявлення та усунення вразливостей.

Business Email Compromise (BEC) — Тип кіберзагрози, що полягає у використанні соціальної інженерії або викрадення електронної пошти для шахрайських дій, таких як фінансові махінації.

Pretexting — Метод соціальної інженерії, при якому зловмисник створює вигаданий сценарій (легенду), щоб отримати доступ до конфіденційної інформації.

Deepfakes — Технологія, що використовує алгоритми штучного інтелекту для створення фальшивих аудіо- або відеозаписів, які імітують реальних осіб.

ШІ (штучний інтелект) — Інтелектуальні системи, що використовують алгоритми для аналізу даних, прийняття рішень або виконання автоматизованих завдань.

Whaling атака — Вид фішингової атаки, спрямованої на високопосадовців компанії (керівників, директорів) з метою отримання конфіденційної інформації або фінансових ресурсів.

Фішинг (Phishing) — Вид кіберзагрози, що полягає у надсиланні підроблених повідомлень або створенні підроблених вебсайтів для викрадення конфіденційної інформації (паролів, номерів кредитних карток).

SSH (Secure Shell) — Протокол безпечного віддаленого доступу до серверів або пристроїв через зашифроване з'єднання.

scp (Secure Copy Protocol) — Протокол для безпечного копіювання файлів між локальними та віддаленими пристроями через SSH.

Сканування портів — Процес перевірки портів на наявність активних сервісів і відкритих точок доступу в системі.

TLS/SSL — Протоколи захисту даних, які забезпечують шифрування з'єднань у мережах, зокрема в інтернеті, для захисту від несанкціонованого доступу.

Проксі-сервер — Проміжний сервер, який виконує функції посередника між користувачем і кінцевим ресурсом для забезпечення анонімності або фільтрації трафіку.

Brute Force — Метод атаки, що полягає у підборі пароля або ключа шифрування шляхом перебору можливих варіантів.

Reverse Shell — Метод атаки, коли зловмисник отримує доступ до пристрою жертви через зворотне з’єднання.

Інформаційна безпека (Information Security) — Сукупність заходів для захисту інформації та систем від несанкціонованого доступу, використання або модифікації.

ВСТУП

Тестування на проникнення (пентест) є важливою складовою забезпечення кібербезпеки в умовах сучасних кіберзагроз. З розвитком цифрових технологій і збільшенням кількості пристроїв, підключених до мережі, виникає необхідність у створенні нових методів і підходів для забезпечення безпеки інформаційних систем. Пентест дозволяє виявити вразливі місця в мережах та інформаційних системах до того, як ними скористаються зловмисники, тим самим знижуючи ризики кібератак.

Raspberry Pi є популярним вибором для виконання пентестів завдяки його компактності, низькій вартості та можливості налаштування для різних обчислювальних задач. У поєднанні з інструментами для сканування, такими як Nmap або OpenVAS, Raspberry Pi дозволяє проводити ефективне тестування мережі, виявляючи вразливі місця, що потребують уваги. Також важливим є забезпечення захищеного доступу до системи за допомогою VPN, що гарантує безпечне управління процесом тестування навіть з віддалених локацій.

Однією з ключових цілей даної магістерської роботи є створення системи, яка використовує Telegram бот для віддаленого керування пентестом з Raspberry Pi та OpenVPN для забезпечення захищеного підключення. Цей підхід дозволяє інтегрувати мобільність та автоматизацію з мінімальними витратами на обладнання, що робить систему доступною як для малого бізнесу, так і для окремих фахівців з кібербезпеки.

У рамках даної магістерської роботи розробляється і досліджується система для безпечного пентесту на базі Raspberry Pi, яка включає налаштування вразливої машини для тестування мережевих вразливостей з використанням VirtualBox та Metasploitable 3. Використання відкритих програмних рішень, таких як OpenVPN і Telegram бот, дозволяє знизити витрати на впровадження та забезпечити гнучке налаштування під конкретні вимоги.

Таким чином, актуальність теми визначається потребою у створенні мобільних, доступних та ефективних рішень для забезпечення кібербезпеки, які можуть застосовуватися в умовах обмежених ресурсів, що особливо важливо в умовах воєнних дій та підвищеної активності кіберзагроз.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд сучасних кіберзагроз та атак

1. Соціальна інженерія

Соціальна інженерія залишається однією з найнебезпечніших та найпоширеніших методик, які використовують кіберзлочинці для доступу до конфіденційної інформації. Цей вид атаки базується на маніпулюванні людською психологією, а не на експлуатації технічних вразливостей. Кіберзлочинці використовують довіру, страх, цікавість або незнання людей, щоб отримати доступ до даних, які вони шукають.

Згідно зі звітом компанії Verizon "2023 Data Breach Investigations Report" [1], приголомшливі 74% усіх витоків даних були пов'язані з певною формою людської взаємодії. Більше того, від 75% до 91% цілеспрямованих кібератак починаються саме з електронної пошти, що підкреслює важливість цього каналу для зловмисників.

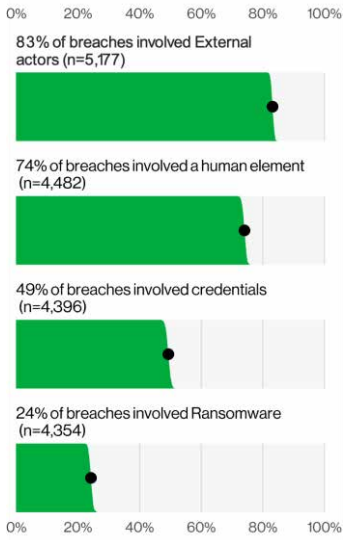


Рисунок 1.1 – Графік, що показує, що 74% усіх порушень включають людський фактор

Важливо зазначити, що соціальна інженерія є надзвичайно ефективною та прибутковою для кіберзлочинців. Можливо, саме тому атаки на ділову електронну пошту (Business Email Compromise, BEC), які по суті є атаками на основі

попереднього обману (pretexting), майже подвоїлися в загальному наборі інцидентів і тепер представляють понад 50% інцидентів у шаблоні соціальної інженерії.

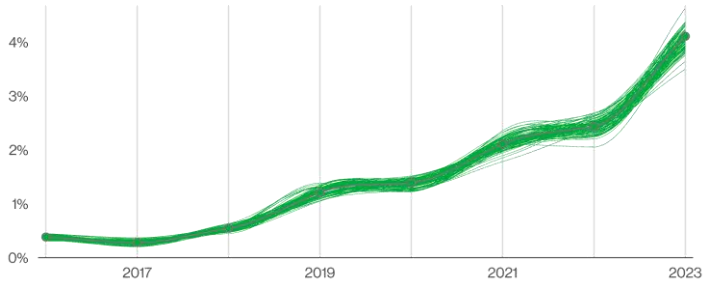


Рисунок 1.2 – Графік зростання атак BEC у загальному наборі інцидентів

Соціальна інженерія стає все більш складною, особливо з розвитком технологій, таких як глибинні підробки (deepfakes) та генеративний штучний інтелект. Це робить атаки важчими для виявлення та запобігання. Компанії з кібербезпеки змушені швидко адаптуватися, розробляючи нові методи та інструменти для протидії цим загрозам.

Поширені типи соціальної інженерії:

- **Фішинг:** Зловмисники надсилають підроблені електронні листи, текстові повідомлення або повідомлення в соціальних мережах, видаючи себе за надійні джерела. Метою є обманом змусити людей розкрити конфіденційну інформацію, таку як паролі, номери кредитних карток або інші особисті дані.
- **Спуфінг:** Зловмисники підробляють адресу електронної пошти або навіть створюють фальшиві веб-сайти, які виглядають майже ідентично справжнім. Це вводить користувачів в оману, змушуючи їх надавати конфіденційну інформацію.
- **Атака на китів (Whaling attack):** Цілеспрямовані атаки на високопоставлених керівників або осіб з доступом до критично важливої інформації. Мета полягає в отриманні доступу до секретних даних або в переконанні жертви здійснити великий грошовий переказ.

- **Атака приманки (Baiting attack):** Зловмисники використовують привабливі пропозиції, такі як безкоштовні продукти або знижки, щоб заманити користувачів на підроблені веб-сайти або змусити їх завантажити шкідливе програмне забезпечення.

Також можна додати, що 83% порушень включають зовнішні дії, і основною мотивацією для атак залишається фінансова вигода, яка становить 95% порушень.

Три основні способи, якими зловмисники отримують доступ до організації:

- Викрадені облікові дані
- Фішинг
- Експлуатація вразливостей

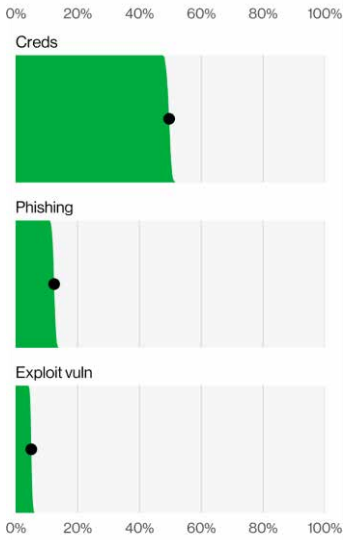


Рисунок 1.2 – Графік, що ілюструє три основні способи доступу зловмисників до організацій

2. Вплив третіх сторін

Кіберзлочинці часто обходять потужні заходи безпеки великих компаній, зламучи менш захищені мережі третіх сторін — постачальників, партнерів або підрядників, які мають привілейований доступ до основної цілі. Цей тип атаки дозволяє зловмисникам проникнути в захищені системи, використовуючи слабкі місця в безпеці інших організацій.

Наприклад, у 2024 році компанія AT&T зіткнулася з масштабним витоком даних, який торкнувся понад 70 мільйонів клієнтів [2]. Витік стався через

стороннього постачальника, що підкреслює небезпеку, яку становлять треті сторони з недостатніми заходами безпеки.

Варто зазначити, що в 2023 році 29% усіх порушень даних сталися через атаки на треті сторони. Це свідчить про те, що компаніям необхідно не лише піклуватися про власну безпеку, але й контролювати безпеку своїх партнерів та постачальників.

3. Помилки конфігурації

Навіть найсучасніші системи безпеки можуть бути вразливими через людські помилки під час їх налаштування. Невеликі помилки в конфігурації можуть призвести до значних вразливостей, які відкривають двері для кіберзлочинців. Згідно зі звітом компанії Censys за 2023 рік, понад 8000 серверів були вразливі до витоку даних через неправильну конфігурацію [3]. Це показує, наскільки важливо ретельно та професійно налаштовувати системи безпеки.

Поширені проблеми конфігурації включають:

- **Використання налаштувань за замовчуванням:** Пристрої, такі як принтери або факси, часто мають стандартні паролі та налаштування, які легко вгадати або знайти. Необхідно завжди змінювати ці налаштування на більш безпечні варіанти.
- **Недостатня сегментація мережі:** Відсутність розділення мережі на сегменти може призвести до того, що зловмисник, отримавши доступ до однієї частини мережі, зможе проникнути в інші, більш конфіденційні області.
- **Невчасне оновлення програмного забезпечення:** Неоновлене програмне забезпечення може мати відомі вразливості, які зловмисники можуть експлуатувати. Регулярні оновлення та виправлення критично важливі для безпеки.
- **Слабкі паролі:** Використання простих або повторюваних паролів значно підвищує ризик несанкціонованого доступу. Компанії повинні встановлювати суворі вимоги до складності паролів та їх регулярної зміни.

4. Кіберзагрози штучного інтелекту

Штучний інтелект (ШІ) кардинально змінив ландшафт кібербезпеки, створюючи нові виклики та загрози. Зловмисники використовують можливості ШІ для автоматизації атак, швидкого аналізу систем безпеки та виявлення вразливостей, які можна експлуатувати. Це призводить до того, що атаки стають не лише більш складними та витонченими, але й частішими.

За даними опитування CFO.com за 2023 рік, 85% фахівців з кібербезпеки вважають, що зростання кількості кібератак пов'язане з використанням тактик на основі ШІ. Крім того, у звіті про індекс кіберризиків за 2023 рік 90% засновників стартапів висловили занепокоєння щодо потенційної небезпеки кібератак, які використовують ШІ [4][5].

Однак ШІ не є лише інструментом у руках кіберзлочинців. Він також надає потужні можливості для покращення захисту. Системи безпеки, що використовують ШІ, можуть більш ефективно виявляти загрози, автоматизувати реагування та навіть прогнозувати можливі атаки. Нові технології, такі як системи виявлення загроз зі штучним інтелектом від IBM, допомагають компаніям бути на крок попереду зловмисників.

5. Тунелювання DNS

Система доменних імен (DNS) є фундаментальною складовою Інтернету, яка відповідає за перетворення доменних імен на IP-адреси. Кіберзлочинці використовують цю систему для проведення атак, відомих як тунелювання DNS. Цей метод дозволяє зловмисникам приховано передавати дані, замасковуючи їх під звичайний DNS-трафік.

Тунелювання DNS є особливо небезпечним, оскільки DNS-трафік зазвичай дозволений через брандмауери, і його важко виявити серед легітимного трафіку. Це відкриває можливість для зловмисників обійти заходи безпеки та отримати доступ до конфіденційних даних або встановити шкідливе програмне забезпечення.

6. Внутрішні загрози

Внутрішні загрози виникають, коли атака походить від співробітника, підрядника або іншої особи з доступом до систем компанії. Ці загрози можуть бути навмисними, коли інсайдер навмисно завдає шкоди або краде дані, або ненавмисними, коли особа випадково створює вразливість, наприклад, потрапляючи на фішингову атаку.

Навмисні внутрішні загрози особливо небезпечні, оскільки інсайдери вже мають доступ до систем і можуть обійти багато заходів безпеки. Наприклад, у 2018 році співробітник Tesla, незадоволений відмовою в підвищенні, навмисно передав третім сторонам значну кількість конфіденційних даних компанії.

7. Атаки, спонсоровані державою

Атаки, спонсоровані державою, є одними з найсерйозніших кіберзагроз. Вони здійснюються державами або урядовими організаціями проти інших країн або організацій з різними цілями: шпигунство, саботаж, розповсюдження дезінформації тощо.

Такі атаки стають все більш поширеними через зростання геополітичної напруженості. Наприклад, у 2024 році було виявлено, що спонсорована Китаєм хакерська група Volt Typhoon атакувала критично важливу інфраструктуру США. Подібні атаки мають серйозні наслідки для національної безпеки всіх країн та їх економіки.

8. Програми-вимагачі

Програми-вимагачі залишаються однією з найнебезпечніших та фінансово руйнівних кіберзагроз сучасності. Цей тип шкідливого програмного забезпечення блокує доступ до комп'ютерних систем або файлів, вимагаючи від жертви сплати викупу за відновлення доступу. У 2024 році ситуація з програмами-вимагачами стала ще більш загрозливою, оскільки кіберзлочинці збільшили як частоту атак, так і розмір вимог викупу. Додатково, їхні методи стали більш витонченими, що ускладнює захист для багатьох організацій.

Згідно зі звітом компанії Sophos "The State of Ransomware 2024", 59% організацій зазнали атак програм-вимагачів протягом останнього року [6]. Хоча це

невелике зниження порівняно з 66% у попередніх двох роках, рівень залишається надзвичайно високим. Це означає, що більше половини всіх компаній у світі були піддані атакам, що свідчить про масштабність проблеми.

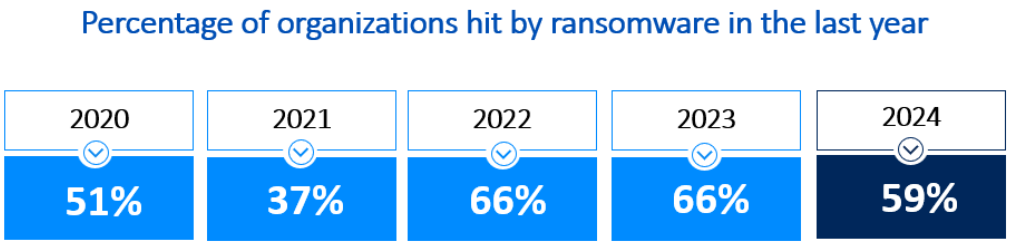


Рисунок 1.3 – Відсоток організацій які зазнали атак програм-вимагачів з 2020 по 2024 рік.

Атаки програм-вимагачів впливають на значну частину інфраструктури компаній. У середньому, 49% комп'ютерів організації зазнають впливу під час такої атаки. Важливо зазначити, що повне шифрування всієї інфраструктури є рідкісним явищем: лише 4% організацій повідомили, що більше 91% їхніх пристроїв були зашифровані. Це свідчить про те, що хоча атаки і масштабні, повні зупинення всіх систем трапляються нечасто.

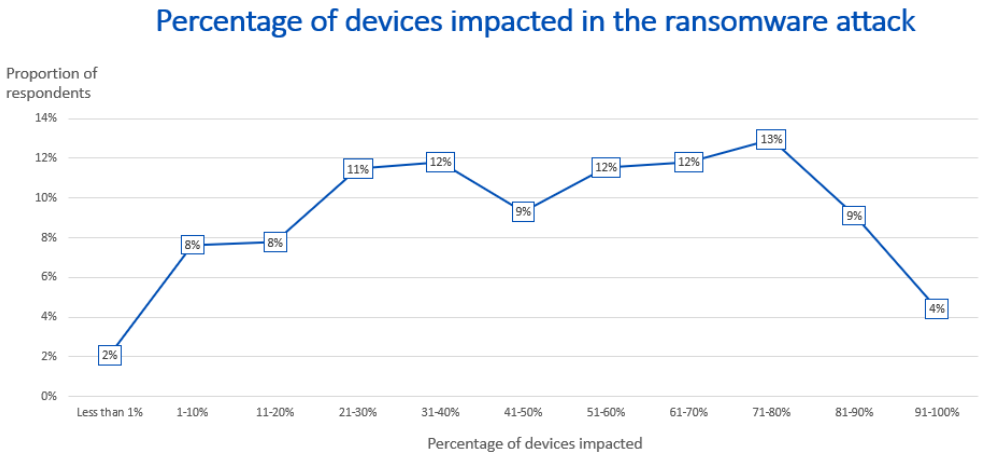


Рисунок 1.4 – Відсоток пристроїв які були зашифровані під час атак

Після атаки програми-вимагача компанії стикаються з важливим питанням, як відновити зашифровані дані.

За даними звіту:

- 68% організацій використовували резервні копії для відновлення даних.
- 56% заплатили викуп, щоб отримати ключ дешифрування.

- 26% використали інші засоби, такі як співпраця з правоохоронними органами або використання публічно доступних ключів дешифрування.

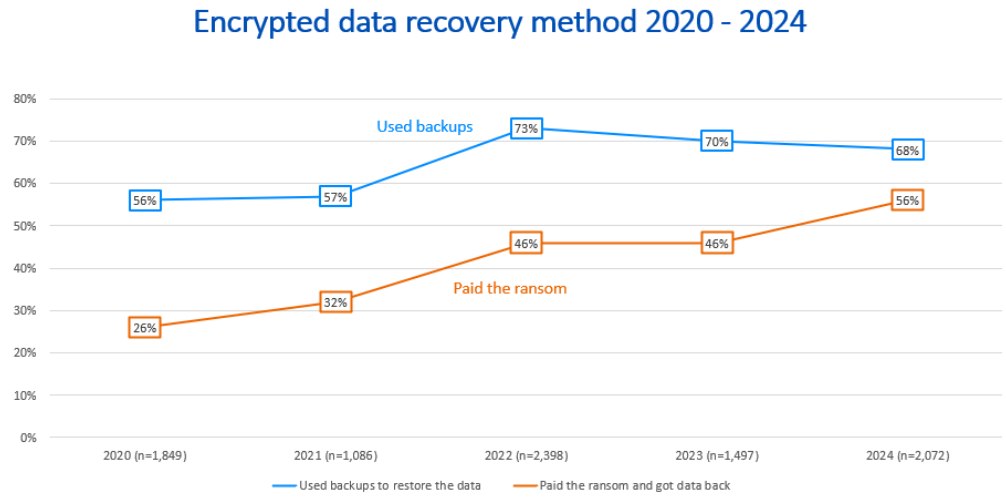


Рисунок 1.5 – Графік методів відновлення зашифрованих даних з 2020 по 2024 рік.

Примітно, що вперше більше половини організацій вирішили заплатити викуп. Це свідчить про те, що кіберзлочинці стали більш успішними у переконанні жертв сплачувати вимоги. Крім того, 47% організацій використовували комбінований підхід, поєднуючи різні методи відновлення. Це більш ніж удвічі більше порівняно з 21% у 2023 році. Така тенденція може вказувати на те, що компанії прагнуть максимально швидко відновити операції, використовуючи всі доступні ресурси.

Кіберзлочинці значно збільшили розмір вимог викупу:

- 63% вимог становили $1 мільйон або більше.
- 30% перевищували $5 мільйонів.

Проте, реальні платежі часто відрізняються від початкових вимог. Лише 24% організацій сплатили суму, яка точно відповідала вимозі. 44% заплатили менше, тоді як 31% заплатили більше, ніж спочатку вимагалося.

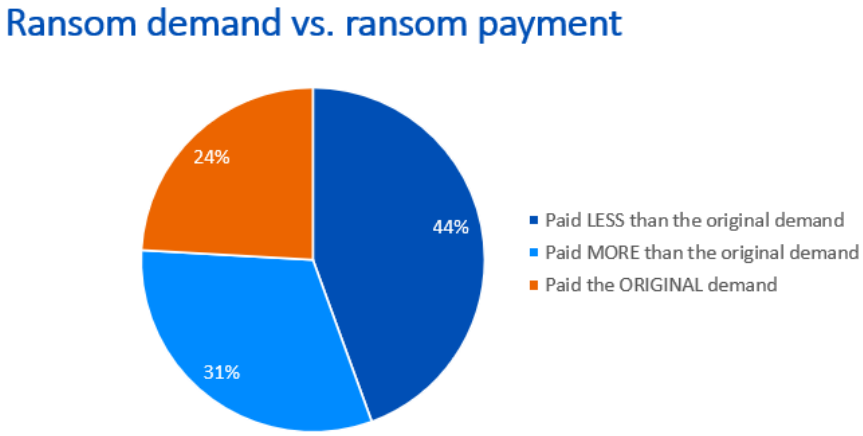


Рисунок 1.6 – Графік порівняння вимог викупу та реальних платежів.

Середній розмір фактично сплаченого викупу збільшився в п'ять разів, з $400 000 у 2023 році до $2 мільйонів у 2024 році. Це свідчить про те, що кіберзлочинці стали успішнішими у отриманні більших сум від жертв.

Рішення про сплату викупу залежить від багатьох факторів:

- Розмір компанії: Великі організації частіше сплачують викуп, оскільки мають більше ресурсів та можуть зазнати більших втрат від простою.
- Наявність резервних копій: Компанії без актуальних резервних копій мають менше варіантів для відновлення даних без сплати викупу.
- Тиск часу: Потреба швидко відновити операції може спонукати до сплати викупу.
- Репутаційні ризики: Побоювання щодо втрати довіри клієнтів або партнерів можуть вплинути на рішення.

Фінансування викупних платежів часто походить з декількох джерел одночасно:

- 40% коштів надають самі організації.
- 23% забезпечуються страховими компаніями.
- 19% фінансуються материнськими компаніями або керівними органами.
- 18% покриваються з особистих коштів фізичних осіб, що може включати керівників або інвесторів.

Ці дані підкреслюють важливість наявності страховки від кібератак, але також вказують на те, що вона рідко покриває всю суму викупу.

Окрім самих платежів викупу, загальні витрати на відновлення після атаки програм-вимагача також зросли:

- Середні витрати на відновлення (без урахування викупу) зросли до $2,73 мільйона, що на $1 мільйон більше, ніж у попередньому році.
- Час відновлення також збільшився: лише 35% організацій повністю відновилися протягом тижня, порівняно з 47% у 2023 році.

Це означає, що атаки стають більш руйнівними, а процес відновлення — більш тривалим та дорогим.

Кіберзлочинці використовують різні методи для запуску атак програм-вимагачів:

- Експлуатація вразливостей: Найпоширеніший метод, який становить 32% інцидентів. Зловмисники використовують незакриті вразливості в програмному забезпеченні для проникнення в систему.
- Компрометовані облікові дані: 29% атак починаються з використання вкрадених або вгаданих облікових даних.
- Шкідливі електронні листи: 23% атак здійснюються через надсилання шкідливих вкладень або посилань.
- Фішинг: 11% атак базуються на обмані користувачів з метою отримання конфіденційної інформації.

Організації, що стали жертвами атак через експлуатацію вразливостей, зазнають більш серйозних наслідків:

- 75% мали скомпрометовані резервні копії.
- 67% мали зашифровані дані.
- 71% заплатили викуп.

Загальні витрати на відновлення для таких організацій становлять $3 мільйони, що вчетверо більше, ніж для тих, хто зазнав атак через компрометовані облікові дані.

Кіберзлочинці все частіше намагаються знищити або скомпрометувати резервні копії, щоб ускладнити відновлення без сплати викупу:

- У 94% випадків зловмисники намагалися отримати доступ до резервних копій.
- 57% цих спроб були успішними.

Організації, чиї резервні копії були скомпрометовані:

- Отримали вдвічі більші вимоги викупу.
- Мали значно більші витрати на відновлення.
- Частіше сплачували викуп.

Крім того, 32% атак включали крадіжку даних, що дозволяє зловмисникам додатково шантажувати жертв, погрожуючи оприлюднити конфіденційну інформацію.

9. Троянські коні

Троянські коні — це шкідливі програми, які маскуються під легітимне програмне забезпечення. Користувачі завантажують та встановлюють їх, не підозрюючи про загрозу. Після встановлення троянський кінь може надавати зловмисникам доступ до системи, красти дані або встановлювати інше шкідливе програмне забезпечення.

Види троянських коней:

- **Backdoor Trojans:** Надають зловмисникам віддалений контроль над комп'ютером.
- **Downloader Trojans:** Завантажують та встановлюють інші шкідливі програми.
- **Ransom Trojans:** Блокують доступ до системи, вимагаючи викуп.
- **Mailfinder Trojans:** Крадуть електронні адреси для розповсюдження спаму.

10. Drive-By Кібератаки

Drive-by атаки відбуваються, коли користувач відвідує скомпрометований веб-сайт, який автоматично завантажує шкідливе програмне забезпечення на

пристрій без відома користувача. Зловмисники можуть впроваджувати шкідливий код на законні веб-сайти, користуючись їх вразливостями.

Такі атаки особливо небезпечні, оскільки не вимагають від користувача жодних дій, окрім відвідування сайту. Встановлення блокувальників реклами та спливаючих вікон, а також уникнення підозрілих веб-сайтів може допомогти запобігти таким атакам.

11. Погана кібергігієна

Найкращий спосіб уникнути кіберзагроз — це освіта та профілактика. Для будь-якої компанії важливо мати хорошу "кібергігієну", яка включає регулярні звички та практики щодо використання технологій.

Приклади належної кібергігієни:

- Уникнення незахищених мереж Wi-Fi
- Використання віртуальних приватних мереж (VPN)
- Реалізація багатофакторної автентифікації
- Створення та дотримання суворих критеріїв паролів
- Шифрування даних
- Оптимізація та правильне налаштування брандмауера
- Регулярне оновлення програмного забезпечення
- Обмеження доступу співробітників до даних
- Використання менеджерів паролів для безпечного зберігання паролів

Дослідження показують, що звички кібергігієни в багатьох організаціях залишають бажати кращого. Наприклад, майже 41% організацій покладаються на людську пам'ять для керування паролями, а 30% записують паролі на папері. Приблизно дві третини американських компаній не використовують менеджери паролів для зберігання паролів.

Крім того, більше половини (54%) IT-спеціалістів не вимагають використання двофакторної автентифікації для доступу до облікових записів компанії, і лише 5% компаній мають кіберексперта у своїй раді директорів.

Зі збільшенням віддаленої роботи доступ до систем, захищених слабкими паролями, тепер здійснюється з незахищених домашніх мереж або навіть відкритих мереж у кафе. Віддалені працівники також можуть використовувати для роботи свої особисті пристрої, які не мають тих самих заходів безпеки та шифрування.

12. Хмарні вразливості

"Хмара" кардинально змінила та покращила цифровий світ, і багато в чому хмарні дані можуть бути безпечнішими, ніж локальні сервери. Тим не менш, часто трапляється навпаки. Згідно з компанією Check Point, лише за останній рік рівень вразливостей у хмарі зріс на 154%.

Таким чином, хоча теоретично хмарні обчислення надзвичайно безпечні, одна невелика неправильна конфігурація або вразливість може призвести до значного витоку даних.

Наприклад, у 2023 році компанія Toyota зазнала масштабного витоку даних, який торкнувся 260 000 клієнтів через неправильну конфігурацію хмари.

Ще один приклад — злам хмари, який вплинув на мільйони клієнтів AT&T, коли хакери отримали доступ до вразливості стороннього хмарного сервісу Snowflake. Ця атака торкнулася майже всіх клієнтів AT&T, що зробило її одним із найбільших витоків даних за всю історію.

13. Уразливості мобільних пристроїв

У сучасному світі мобільні пристрої стали невід'ємною частиною нашого життя. Вони містять величезну кількість особистої та корпоративної інформації. Це робить їх привабливою ціллю для кіберзлочинців.

Мобільні пристрої часто не мають таких же заходів безпеки, як комп'ютери, наприклад брандмауери або антивірусне програмне забезпечення. Це робить їх більш вразливими до атак. Зловмисники можуть використовувати різні методи, такі як шкідливі додатки, фішингові повідомлення або експлуатація вразливостей операційної системи.

Крім того, багато людей використовують свої особисті мобільні пристрої для роботи, що може створювати ризики для корпоративної безпеки. Без належних

політик та інструментів управління мобільними пристроями компанії можуть бути вразливими до витоку даних.

14. Інтернет речей (IoT)

Інтернет речей (IoT) — це мережа взаємопов'язаних пристроїв, які можуть збирати, обробляти та обмінюватися даними через Інтернет. Ці пристрої включають розумні телевізори, термостати, охоронні системи, носимі пристрої, розумні годинники, автомобілі з підключенням до мережі, медичне обладнання та багато іншого. IoT-пристрої широко використовуються як у побуті, так і в промисловості, охоплюючи сфери охорони здоров'я, енергетики, транспорту та виробництва.

Зростання популярності та вразливості

З розвитком технологій та зниженням вартості підключення до Інтернету кількість IoT-пристроїв швидко зростає. За прогнозами аналітичних компаній, до 2025 року кількість підключених пристроїв може перевищити **75 мільярдів**. Однак разом зі зростанням популярності збільшується і кількість потенційних вразливостей, які можуть бути використані кіберзлочинцями.

Багато IoT-пристроїв мають обмежені можливості обробки та зберігання даних, що ускладнює впровадження стандартних заходів безпеки. Крім того, виробники часто приділяють більше уваги функціональності та швидкому виходу на ринок, ніж безпеці, що призводить до випуску пристроїв із слабкими або взагалі відсутніми заходами захисту.

Поширені вразливості IoT-пристроїв

- **Слабкі або стандартні паролі**: Багато пристроїв постачаються зі стандартними паролями, які користувачі часто не змінюють, що робить їх легкою мішенню для зловмисників.
- **Відсутність шифрування**: Дані, що передаються між IoT-пристроями та серверами, можуть бути незашифрованими, що дозволяє перехоплювати та змінювати інформацію.

- **Невчасне оновлення програмного забезпечення**: Багато пристроїв не отримують регулярних оновлень безпеки, залишаючи відомі вразливості незакритими.
- **Відсутність стандартизації**: Різні виробники використовують різні протоколи та стандарти, що ускладнює забезпечення єдиних заходів безпеки.

Приклади кібератак на IoT-пристрої

- **Mirai Botnet (2016)**: Одна з найбільших DDoS-атак в історії, яка використовувала тисячі скомпрометованих IoT-пристроїв, включаючи камери спостереження та маршрутизатори, для перевантаження сервісів на зразок Dyn DNS.
- **Атаки на медичне обладнання**: Зловмисники можуть отримати доступ до медичних пристроїв, таких як інсулінові помпи або кардіостимулятори, що може поставити під загрозу життя пацієнтів.
- **Злам розумних будинків**: Хакери можуть отримати контроль над системами безпеки, освітленням, опаленням та іншими компонентами розумних будинків, створюючи ризики для безпеки мешканців.

Статистичні дані

У 2022 році було зареєстровано понад **112 мільйонів кібератак** на пристрої IoT, порівняно з **32 мільйонами** у 2018 році. Це свідчить про те, що кіберзлочинці все більше націлюються на ці пристрої, використовуючи їх як точку входу до мереж або для створення ботнетів. Збільшення кількості атак підкреслює необхідність посилення заходів безпеки для IoT-пристроїв.

Наслідки для безпеки

- **Порушення приватності**: Збір та передача особистих даних без належного захисту можуть призвести до витоку конфіденційної інформації.
- **Компрометація мережі**: Скомпрометовані IoT-пристрої можуть використовуватися як плацдарм для атак на інші пристрої в мережі.
- **Фізична безпека**: Управління фізичними об'єктами, такими як автомобілі або промислове обладнання, може призвести до матеріальних збитків або шкоди для людей.

15.Неналежне Управління Даними

Управління даними є критично важливим аспектом успішного функціонування сучасних організацій. Воно включає не лише систематизацію та зберігання інформації, але й забезпечення її безпеки, доступності та цілісності. У цифрову епоху обсяги даних зростають експоненціально, створюючи як можливості для бізнесу, так і нові виклики в сфері кібербезпеки.

Експоненціальний ріст даних

За останні роки спостерігається стрімке зростання обсягу даних, які створюються та зберігаються в цифровому вигляді. За оцінками експертів, кількість даних, створених споживачами, **подвоюється кожні два роки**. Це означає, що організації повинні обробляти та зберігати величезні обсяги інформації, що вимагає ефективних стратегій управління.

Наприклад, у 2020 році загальний обсяг даних, створених, захоплених, скопійованих і спожитих у світі, досягнув приблизно **64 зетабайт** (1 зетабайт = 1 трильйон гігабайт). Очікується, що до 2025 року цей обсяг зросте до **175 зетабайт**.

Виклики поганого управління даними

Накопичення надлишкових та невпорядкованих даних може призвести до ряду проблем:

- **Плутанина та неефективність**: Велика кількість неструктурованих даних ускладнює доступ до необхідної інформації, що може сповільнити бізнес-процеси та прийняття рішень.
- **Підвищений ризик кіберзагроз**: Невпорядковані дані часто не захищені належним чином, що робить їх вразливими до кібератак, таких як несанкціонований доступ, витік даних або крадіжка конфіденційної інформації.
- **Порушення законодавства**: Невиконання вимог щодо зберігання та захисту даних може призвести до юридичних наслідків, включаючи штрафи та втрату репутації.

Регуляторні вимоги та відповідальність

У багатьох країнах існують суворі закони та регуляторні акти, які зобов'язують організації забезпечувати належний захист даних. Наприклад, у США **Федеральна торгова комісія (FTC)** регулює безпеку даних та може накладати штрафи на компанії, які не забезпечують захист конфіденційної інформації.

В Україні також діють законодавчі акти, такі як **Закон України "Про захист персональних даних"**, який зобов'язує організації дотримуватися вимог щодо збору, обробки та зберігання персональних даних.

Стратегічне управління даними

Для уникнення негативних наслідків, пов'язаних з поганим управлінням даними, організаціям рекомендується розробити та впровадити **стратегічні плани управління даними**, які включають:

- **Інвентаризацію даних**: Визначення, які дані зберігаються, де вони знаходяться та хто має до них доступ.
- **Класифікацію даних**: Розподіл даних за рівнем важливості та конфіденційності для встановлення відповідних заходів захисту.
- **Розробку політик та процедур**: Встановлення чітких правил щодо збору, обробки, зберігання та видалення даних.
- **Впровадження технологічних рішень**: Використання сучасних інструментів для управління даними, таких як системи управління базами даних, засоби резервного копіювання та шифрування.
- **Навчання персоналу**: Регулярне навчання співробітників щодо важливості захисту даних та дотримання політик безпеки.

Переваги належного управління даними

Ефективне управління даними надає організаціям ряд переваг:

- **Підвищення ефективності**: Легкий доступ до структурованих даних сприяє швидшому прийняттю рішень та оптимізації бізнес-процесів.
- **Покращення безпеки**: Належно захищені дані менш вразливі до кібератак та витоків.
- **Відповідність законодавству**: Дотримання регуляторних вимог допомагає уникнути штрафів та юридичних проблем.

- **Збереження репутації**: Захист даних клієнтів та партнерів зміцнює довіру до організації.

Реальні приклади наслідків поганого управління даними

Багато компаній зазнали серйозних втрат через недостатню увагу до управління даними. Наприклад:

- **Витоки даних**: Неналежне зберігання конфіденційної інформації призвело до великих витоків даних у таких компаніях, як Equifax та Yahoo, що спричинило значні фінансові втрати та шкоду репутації.
- **Штрафи за невідповідність**: Європейський Союз накладає значні штрафи за порушення **Загального регламенту захисту даних (GDPR)**. Компанії, які не дотримуються вимог щодо управління даними, можуть бути оштрафовані на суми до **€20 мільйонів** або **4% світового річного обороту**.

1.2 Використання Raspberry Pi в кібербезпеці: переваги, обмеження та порівняння з альтернативами

Raspberry Pi — це невеликий, доступний та високофункціональний одноплатний комп'ютер, який став популярним серед ентузіастів та професіоналів у сфері інформаційних технологій. Завдяки своїй універсальності та низькій вартості, Raspberry Pi знайшов широке застосування в різних галузях, включаючи кібербезпеку.

Огляд використання Raspberry Pi в кібербезпеці

Raspberry Pi може бути використаний як потужний інструмент для тестування на проникнення, моніторингу мережі, створення honeypot-систем та інших завдань, пов'язаних з кібербезпекою. Завдяки можливості встановлення різних операційних систем, включаючи спеціалізовані дистрибутиви для кібербезпеки, такі як Kali Linux, Raspberry Pi стає універсальним інструментом для фахівців з безпеки.

Приклади можливих проєктів на Raspberry Pi для кібербезпеки

1. **Portable Penetration Testing Device**

Raspberry Pi може бути перетворений на портативний пристрій для тестування на проникнення. Встановивши Kali Linux та набір необхідних інструментів, фахівці можуть використовувати Raspberry Pi для проведення аудиту безпеки мереж, виявлення вразливостей та тестування на проникнення.

2. **Моніторинг мережевого трафіку**

Використовуючи Raspberry Pi в якості мережевого сенсора, можна здійснювати моніторинг мережевого трафіку для виявлення аномалій та потенційних атак. Інструменти на зразок Snort або Suricata можуть бути встановлені на Raspberry Pi для реалізації системи виявлення вторгнень (IDS).

3. **Honeypot-системи**

Raspberry Pi може бути використаний для створення honeypot-систем — пасток для зловмисників. Розгорнувши на ньому такі інструменти, як Cowrie або

Dionaea, можна збирати інформацію про атаки та зловмисні дії в мережі, що допомагає покращити захист.

4. **Виявлення шкідливого програмного забезпечення**

Raspberry Pi може бути використаний для аналізу шкідливого програмного забезпечення в ізольованому середовищі. Завдяки низькій вартості, можна створити цілу лабораторію для дослідження вірусів та інших загроз.

5. **Бездротові атаки та аудит безпеки Wi-Fi**

З додатковими модулями, Raspberry Pi може бути використаний для проведення аудиту безпеки бездротових мереж. Інструменти, такі як Aircrack-ng, дозволяють перевіряти мережі на вразливості, проводити тестування на проникнення та виявляти несанкціонований доступ.

6. **Автоматизація резервного копіювання**

Raspberry Pi може служити як сервер для автоматизації процесу резервного копіювання важливих даних. Це допомагає забезпечити збереження інформації та швидке відновлення у разі кібератаки або системного збою.

7. **Фізичне управління доступом**

Поєднуючи Raspberry Pi з RFID-читачами та іншими датчиками, можна створювати системи контролю фізичного доступу до приміщень або обладнання, підвищуючи безпеку організації.

Переваги використання Raspberry Pi в кібербезпеці

1. **Доступність та низька вартість**

Raspberry Pi є одним з найдоступніших комп'ютерів, що дозволяє створювати масштабовані рішення без значних фінансових витрат.

2. **Компактність та портативність**

Маленькі розміри Raspberry Pi дозволяють використовувати його в місцях з обмеженим простором або створювати мобільні рішення для тестування та моніторингу.

3. **Гнучкість та універсальність**

Можливість встановлення різних операційних систем та програмного забезпечення робить Raspberry Pi універсальним інструментом для різних завдань у сфері кібербезпеки.

4. **Велика спільнота та підтримка**

Завдяки великій спільноті користувачів та розробників, існує безліч ресурсів, навчальних матеріалів та готових рішень, які можна використовувати або адаптувати під свої потреби.

5. **Енергоефективність**

Низьке енергоспоживання Raspberry Pi робить його ідеальним для безперервної роботи, що важливо для моніторингу та інших безперервних завдань.

Обмеження використання Raspberry Pi в кібербезпеці

1. **Обмежена обчислювальна потужність**

Порівняно з повноцінними комп'ютерами або серверами, Raspberry Pi має обмежену продуктивність, що може бути критичним при виконанні ресурсомістких завдань.

2. **Обмежена пам'ять**

Невеликий обсяг оперативної пам'яті може обмежувати можливості паралельного виконання декількох додатків або обробки великих обсягів даних.

3. **Відсутність деяких апаратних інтерфейсів**

Raspberry Pi може не підтримувати певні апаратні інтерфейси або мати обмежені можливості підключення, що може бути важливим для деяких проєктів.

4. **Потреба в додатковій безпеці**

Як і будь-який інший пристрій, Raspberry Pi потребує належного налаштування безпеки, оскільки сам може стати ціллю атак.

5. **Відсутність апаратної підтримки шифрування**

На відміну від деяких спеціалізованих пристроїв, Raspberry Pi не має вбудованих модулів для апаратного шифрування, що може впливати на швидкість обробки зашифрованих даних.

Порівняння Raspberry Pi з альтернативами в кібербезпеці

Хоча Raspberry Pi є вибором для деяких проєктів у сфері кібербезпеки, існують інші одноплатні комп'ютери та мікроконтролери, які можуть бути використані для подібних цілей. Розглянемо деякі з найпоширеніших альтернатив та порівняємо їх з Raspberry Pi.

1. **Arduino**

Опис:

Arduino — це платформа мікроконтролерів з відкритим вихідним кодом, призначена для створення інтерактивних електронних проєктів.

Переваги:

- **Низька вартість**

  Arduino плати часто дешевші за Raspberry Pi.

- **Простота використання**

  Легко освоїти для початківців, особливо для апаратних проєктів.

- **Велика спільнота та підтримка**

  Широка база користувачів та багато ресурсів.

Обмеження:

- **Обмежені обчислювальні можливості**

Arduino є мікроконтролером, а не повноцінним комп'ютером, і не може виконувати складні програмні завдання.

- **Відсутність операційної системи**

Не підтримує ОС на кшталт Linux, що обмежує можливості використання програмного забезпечення для кібербезпеки.

Висновок:

Arduino підходить для простих апаратних проєктів, але обмежений для завдань кібербезпеки, які потребують більшої обчислювальної потужності та підтримки операційної системи.

2. **BeagleBone Black**

Опис:

BeagleBone Black — це одноплатний комп'ютер, подібний до Raspberry Pi, але з деякими відмінностями в апаратній частині.

Переваги:

- **Більша кількість GPIO-пінів**

Підходить для проєктів, що потребують багато підключень до периферійних пристроїв.

- **Висока продуктивність**

Має дещо вищу обчислювальну потужність у порівнянні з деякими моделями Raspberry Pi.

- **Вбудована eMMC-пам'ять**

Можливість завантаження ОС з внутрішньої пам'яті, що може бути швидшим та надійнішим за SD-карту.

Обмеження:

- **Менша спільнота**

Менше користувачів та ресурсів у порівнянні з Raspberry Pi.

- **Вища вартість**

Дорожчий за Raspberry Pi, що може бути важливим для бюджетних проєктів.

Висновок:

BeagleBone Black підходить для складних апаратних проєктів, але менша спільнота та вища вартість можуть бути перешкодою для широкого використання в кібербезпеці.

3. **Odroid**

Опис:

Odroid — серія одноплатних комп'ютерів від компанії Hardkernel, що пропонує високу продуктивність.

Переваги:

- **Висока обчислювальна потужність**

Деякі моделі Odroid мають потужніші процесори та більше оперативної пам'яті.

- **Підтримка різних операційних систем**

  Можливість встановлення Linux, Android та інших ОС.

  Обмеження:

- **Вища вартість**

  Деякі моделі можуть бути значно дорожчими за Raspberry Pi.

- **Менша спільнота**

  Менше ресурсів та підтримки від спільноти.

  Висновок:

Odroid може бути гарним вибором для проєктів, що потребують високої продуктивності, але вища вартість та менша підтримка можуть обмежувати його використання.

4. **PINE64**

Опис:

PINE64 — серія одноплатних комп'ютерів, які позиціонуються як конкурент Raspberry Pi з відкритою архітектурою.

Переваги:

- **Відкритий дизайн**

  Повністю відкриті специфікації та дизайн.

- **Різноманітність моделей**

  Широкий вибір моделей з різними характеристиками.

  Обмеження:

- **Проблеми з програмною підтримкою**

  Можуть виникати складнощі з драйверами та сумісністю операційних систем.

- **Менша спільнота**

  Менше ресурсів та підтримки в порівнянні з Raspberry Pi.

  Висновок:

PINE64 може бути цікавим для ентузіастів відкритого обладнання, але програмні проблеми можуть обмежувати його використання в кібербезпеці.

Чому Raspberry Pi є кращим вибором для кібербезпеки

Після порівняння Raspberry Pi з альтернативами можна зробити висновок, що він є оптимальним вибором для більшості проєктів у сфері кібербезпеки. Основні причини цього:

- **Баланс ціни та продуктивності**

Raspberry Pi пропонує достатню обчислювальну потужність за доступною ціною, що робить його привабливим для різних проєктів.

- **Підтримка різних операційних систем та програмного забезпечення**

Можливість встановлення спеціалізованих дистрибутивів для кібербезпеки, таких як Kali Linux, значно розширює функціональність пристрою.

- **Велика та активна спільнота**

Широка база користувачів забезпечує доступ до великої кількості ресурсів, документації та готових рішень, що полегшує розробку та вирішення проблем.

- **Широка підтримка периферійних пристроїв та аксесуарів**

Різноманітність доступних аксесуарів та модулів дозволяє адаптувати Raspberry Pi під конкретні потреби, включаючи підключення датчиків, камер, бездротових модулів та інше.

- **Довгострокова підтримка та регулярні оновлення**

Raspberry Pi Foundation постійно випускає оновлення та нові моделі, що забезпечує актуальність та сумісність пристроїв з новим програмним забезпеченням.

Чому Raspberry Pi був обраний для даної магістерської роботи

Враховуючи всі вищезазначені переваги та особливості, **Raspberry Pi** виступає ідеальним інструментом для досліджень та розробки в сфері кібербезпеки в рамках цієї магістерської роботи. Його **доступність** та **низька вартість** дозволяють реалізовувати комплексні проєкти без значних фінансових витрат, що є важливим фактором у академічному середовищі.

Гнучкість та **універсальність** Raspberry Pi надають можливість адаптувати його під специфічні потреби дослідження, встановлювати необхідні операційні системи та програмне забезпечення, зокрема спеціалізовані дистрибутиви для кібербезпеки. Це відкриває широкий спектр можливостей для експериментів та тестування різних підходів до захисту інформаційних систем.

Крім того, **велика та активна спільнота** користувачів та розробників Raspberry Pi забезпечує доступ до численних ресурсів, навчальних матеріалів та готових рішень. Це сприяє швидкому вирішенню технічних питань, обміну досвідом та постійному вдосконаленню навичок, що є невід'ємною частиною успішного проведення наукових досліджень.

Таким чином, **Raspberry Pi** був обраний як оптимальна платформа для дослідження та реалізації проєкту в рамках магістерської роботи, що дозволяє поєднати теоретичні знання з практичними навичками, розвинути компетенції у сфері кібербезпеки та зробити внесок у розвиток цієї важливої галузі.

1.3Telegram-бот як засіб управління системами кібербезпеки

Telegram — популярний месенджер з відкритим API, який дозволяє розробникам створювати ботів — автоматизовані програми, що взаємодіють з користувачами або іншими системами. Завдяки своїй гнучкості та багатофункціональності, Telegram-боти знайшли широке застосування в різних сферах, включаючи кібербезпеку.

Загальні можливості Telegram-ботів

1. **Автоматизація завдань**
   - **Виконання скриптів та програм**: Боти можуть запускати скрипти або програми для автоматизації різних процесів.
   - **Розклад завдань**: Можливість виконання завдань за розкладом або у відповідь на певні події.
2. **Інтерактивна взаємодія з користувачами**
   - **Обробка команд**: Боти можуть розпізнавати та реагувати на команди, надіслані користувачами.
   - **Меню та кнопки**: Використання інтерактивних елементів для спрощення навігації та виконання дій.
   - **Опитування та форми**: Збір даних або зворотного зв'язку від користувачів.
3. **Сповіщення та алерти**
   - **Миттєві повідомлення**: Надсилання оперативних сповіщень про події або зміни в системі.
   - **Налаштування рівнів важливості**: Розподіл повідомлень за пріоритетами або категоріями.
4. **Інтеграція з іншими сервісами та API**
   - **Підключення до зовнішніх API**: Отримання та обробка даних з інших систем.
   - **Взаємодія з базами даних**: Збереження та витяг інформації з баз даних.

5. **Обробка файлів та мультимедіа**
    - **Надсилання та отримання документів**: Обмін файлами різних форматів.
    - **Обробка зображень та відео**: Аналіз або модифікація мультимедійного контенту.
6. **Безпека та контроль доступу**
    - **Аутентифікація користувачів**: Перевірка прав доступу та ідентифікація користувачів.
    - **Шифрування даних**: Забезпечення конфіденційності переданої інформації.
7. **Підтримка групових чатів та каналів**
    - **Взаємодія з групами**: Надсилання сповіщень або інформації в групові чати.
    - **Публікація в каналах**: Розповсюдження інформації для великої аудиторії.

Приклади використання ботів у сфері кібербезпеки

1. Моніторинг та сповіщення про безпекові події

- **Інтеграція з системами моніторингу**: Боти можуть отримувати дані від систем моніторингу (наприклад, SIEM, IDS/IPS) та надсилати сповіщення про інциденти безпеки.
- **Аналіз лог-файлів**: Автоматизований аналіз журналів подій та повідомлення про аномалії.

2. Автоматизація реагування на інциденти

- **Запуск скриптів реагування**: Боти можуть автоматично запускати скрипти для ізоляції скомпрометованих систем або блокування підозрілої активності.
- **Координація дій команди**: Надсилання інструкцій або плану дій членам команди безпеки.

3. Управління доступом та автентифікація

- **Двофакторна автентифікація**: Надсилання одноразових кодів для підтвердження особи користувача.

- **Управління паролями**: Генерація та розповсюдження тимчасових паролів або ключів доступу.

4. Інформаційна підтримка та навчання

- **Розповсюдження новин та оновлень**: Інформування про нові загрози, вразливості та рекомендації з безпеки.
- **Навчальні матеріали**: Надсилання посібників, порад або організація інтерактивних тестів для підвищення обізнаності співробітників.

5. Інтеграція з DevSecOps-процесами

- **Сповіщення про результати безпекового тестування**: Інформування розробників про виявлені вразливості в коді.
- **Автоматизація перевірок**: Запуск статичного або динамічного аналізу безпеки додатків та повідомлення про результати.

6. Управління конфігурацією та оновленнями

- **Контроль версій програмного забезпечення**: Інформування про доступність оновлень та патчів.
- **Автоматизоване розповсюдження оновлень**: Запуск процесу оновлення систем через інтеграцію з інструментами управління конфігурацією.

Порівняння Telegram-ботів з іншими інструментами

1. Інші месенджери з підтримкою ботів

a) Slack

- **Переваги Slack:**
  - Глибока інтеграція з корпоративними інструментами та сервісами.
  - Розширені можливості для командної роботи та співпраці.
- **Недоліки Slack:**
  - Висока вартість підписки для розширених функцій.
  - Обмежена кількість повідомлень у безкоштовній версії.
  - Менша аудиторія поза корпоративним середовищем.

b) Microsoft Teams

- **Переваги Microsoft Teams:**
  - Тісна інтеграція з екосистемою Microsoft 365.

- Підтримка корпоративних стандартів безпеки та управління.

- **Недоліки Microsoft Teams:**
  - Вимога корпоративних ліцензій та підписки.
  - Складність інтеграції з нестандартними або сторонніми системами.
  - Менша гнучкість у розробці ботів порівняно з Telegram.

c) WhatsApp

- **Переваги WhatsApp:**
  - Велика користувацька база по всьому світу.
  - Простота використання та звичність для багатьох користувачів.
- **Недоліки WhatsApp:**
  - Обмежений доступ до API для розробки ботів.
  - Строгі правила та обмеження щодо використання ботів.
  - Недостатня гнучкість для реалізації складних функцій.

Порівняння з Telegram:

- **Відкритий та потужний API**: Telegram надає розробникам більше можливостей для створення ботів з різноманітним функціоналом.
- **Безкоштовність та доступність**: Telegram не має обмежень на використання API та не вимагає оплати за розширені функції.
- **Широка аудиторія**: Telegram використовується як в особистих, так і в корпоративних цілях.
- **Швидкість та надійність**: Миттєва доставка повідомлень та стабільна робота сервісу.

Чому був обраний саме Telegram-бот для цієї роботи

1. Відкритий та потужний API

Telegram надає розробникам потужний та гнучкий API без обмежень та платних підписок. Це дозволяє реалізувати необхідний функціонал бота без технічних бар'єрів.

2. Швидкість розробки та впровадження

Завдяки наявності численних бібліотек та прикладів, розробка Telegram-бота займає відносно небагато часу. Це дозволяє зосередитися на основних завданнях проєкту та швидко отримати працюючий прототип.

3. Доступність та крос-платформеність

Telegram доступний на всіх популярних платформах: Windows, macOS, Linux, Android, iOS та навіть у веб-версії. Це забезпечує зручний доступ до бота з будь-якого пристрою, що є важливим для оперативного управління системами кібербезпеки.

4. Інтерактивність та зручність використання

Можливість двосторонньої взаємодії з ботом у режимі реального часу дозволяє швидко отримувати необхідну інформацію та виконувати дії. Інтуїтивно зрозумілий інтерфейс Telegram спрощує процес взаємодії.

5. Широка аудиторія та популярність

Telegram має велику користувацьку базу, що полегшує інтеграцію бота в існуючі процеси та команди. Це знижує бар'єри для впровадження та навчання персоналу.

6. Безкоштовність використання

Відсутність плати за використання Telegram та його API робить його привабливим вибором для проєктів з обмеженим бюджетом, включаючи академічні дослідження та стартапи.

7. Гнучкість та масштабованість

Telegram-боти легко адаптуються до змінних потреб проєкту. Вони можуть бути розширені новим функціоналом або інтегровані з іншими системами без значних зусиль.

8. Безпека та конфіденційність

Telegram забезпечує шифрування повідомлень та має додаткові функції безпеки, такі як двофакторна автентифікація. Хоча це не повністю усуває всі ризики, але надає додатковий рівень захисту.

9. Спільнота та підтримка

Велика спільнота розробників та користувачів Telegram забезпечує доступ до ресурсів, документації та швидкого вирішення можливих проблем.

1.4 Використання OpenVPN: переваги, обмеження та порівняння з альтернативами

OpenVPN — це потужне, відкрите та безкоштовне програмне забезпечення для створення віртуальних приватних мереж (VPN). Воно дозволяє встановлювати захищені з'єднання між віддаленими комп'ютерами або мережами через незахищені мережі, такі як Інтернет. OpenVPN підтримує різні платформи та використовується як у корпоративному середовищі, так і для особистих потреб.

Можливості та функціональність OpenVPN

1. **Безпека та шифрування**
   - **Підтримка різних протоколів шифрування:** OpenVPN використовує протоколи SSL/TLS для встановлення захищеного з'єднання.
   - **Аутентифікація клієнтів та серверів:** Підтримка сертифікатів X.509, попередньо спільних ключів та облікових даних користувача.
   - **Захист від атак:** Вбудовані механізми для запобігання DoS-атакам, MITM-атакам та іншим загрозам.
2. **Гнучкість та налаштування**
   - **Підтримка різних конфігурацій:** Точка-точка або точка-багатоточка з'єднання.
   - **Налаштування портів та протоколів:** Можливість працювати через TCP або UDP, на різних портах.
   - **Підтримка NAT та брандмауерів:** OpenVPN може проходити через NAT та працювати за жорсткими мережевими обмеженнями.
3. **Крос-платформеність**
   - **Підтримка різних операційних систем:** Windows, macOS, Linux, Android, iOS та інші.
   - **Відкритий код:** Дозволяє модифікувати та адаптувати програмне забезпечення під специфічні потреби.
4. **Масштабованість**

- **Підтримка великої кількості одночасних підключень:** Підходить для як малих, так і великих організацій.
- **Можливість використання в хмарних середовищах:** Легко інтегрується з AWS, Azure, Google Cloud тощо.

Приклади використання OpenVPN у кібербезпеці

1. **Забезпечення безпечного віддаленого доступу**

OpenVPN дозволяє співробітникам безпечно підключатися до корпоративної мережі з будь-якого місця, забезпечуючи захист даних та ресурсів від несанкціонованого доступу.

2. **Об'єднання віддалених офісів та центрів обробки даних**

За допомогою OpenVPN можна створювати захищені канали між різними офісами або дата-центрами, об'єднуючи їх в єдину мережу.

3. **Захист мережевого трафіку в незахищених мережах**

Користувачі можуть використовувати OpenVPN для шифрування свого трафіку при підключенні до публічних Wi-Fi мереж, запобігаючи перехопленню даних.

4. **Реалізація безпечних IoT-рішень**

OpenVPN може бути використаний для захисту з'єднань між пристроями Інтернету речей та серверними системами.

Переваги використання OpenVPN

1. **Високий рівень безпеки**
   - **Сильне шифрування:** Підтримка сучасних алгоритмів шифрування забезпечує захист даних.
   - **Аутентифікація:** Багатофакторна аутентифікація підвищує рівень безпеки доступу.
2. **Гнучкість та налаштування**
   - **Можливість адаптації:** Відкрите програмне забезпечення дозволяє налаштовувати OpenVPN під специфічні потреби організації.
   - **Підтримка різних мережевих топологій:** Підходить для різних сценаріїв використання.

3. **Крос-платформеність та сумісність**
    - **Широка підтримка платформ:** Забезпечує сумісність між різними операційними системами та пристроями.
4. **Економічна ефективність**
    - **Безкоштовне використання:** Відкрита ліцензія дозволяє використовувати OpenVPN без ліцензійних витрат.
    - **Зниження витрат на інфраструктуру:** Відсутність потреби в дорогому обладнанні.

Обмеження використання OpenVPN

1. **Складність налаштування**
    - **Технічні вимоги:** Потребує знань для правильного налаштування та оптимізації.
    - **Можливі помилки конфігурації:** Неправильне налаштування може призвести до вразливостей.
2. **Продуктивність та швидкість**
    - **Навантаження на систему:** Шифрування та дешифрування трафіку можуть впливати на продуктивність, особливо на слабких пристроях.
    - **Затримки в з'єднанні:** Можуть виникати через додаткову обробку трафіку.
3. **Відсутність офіційної підтримки**
    - **Спільнота та документація:** Хоча існує велика спільнота, відсутність офіційної технічної підтримки може бути проблемою для деяких організацій.

Порівняння OpenVPN з іншими VPN-рішеннями

1. IPSec (Internet Protocol Security)

Опис:

IPSec — це набір протоколів для забезпечення захищених комунікацій через мережі IP, часто використовується для створення VPN на рівні мережевого протоколу.

Переваги:

- **Висока безпека на рівні мережі:** Забезпечує захист на рівні IP-пакетів.
- **Широка підтримка:** Інтегрований у багато мережевих пристроїв та операційних систем.

Недоліки:

- **Складність налаштування:** Потребує значних знань для правильного налаштування.
- **Проблеми з NAT:** Може мати труднощі при проходженні через NAT.

Порівняння з OpenVPN:

- **Гнучкість:** OpenVPN більш гнучкий у налаштуванні та може працювати через NAT без проблем.
- **Простота налаштування:** OpenVPN простіший для впровадження в малих та середніх організаціях.

2. WireGuard

Опис:

WireGuard — це сучасний VPN-протокол, розроблений для забезпечення високої швидкості та простоти використання.

Переваги:

- **Висока продуктивність:** Швидше за OpenVPN та IPSec завдяки ефективній реалізації.
- **Простота:** Менше коду та простіша конфігурація.

Недоліки:

- **Молодий проект:** Ще не отримав широкого поширення та перевірки часом.
- **Обмежена функціональність:** Менше можливостей налаштування порівняно з OpenVPN.

Порівняння з OpenVPN:

- **Швидкість:** WireGuard зазвичай швидший за OpenVPN.
- **Зрілість:** OpenVPN існує довше та має більшу спільноту та підтримку.
- **Безпека:** Обидва протоколи безпечні, але OpenVPN більш перевірений у реальних умовах.

3. PPTP (Point-to-Point Tunneling Protocol)

Опис:

PPTP — старий VPN-протокол, який забезпечує базовий рівень захисту.

Переваги:

- **Простота налаштування:** Легкий у встановленні та конфігурації.
- **Широка підтримка:** Підтримується багатьма старими пристроями та операційними системами.

Недоліки:

- **Низький рівень безпеки:** Відомі вразливості роблять його ненадійним для захисту даних.
- **Застарілість:** Більшість сучасних організацій відмовилися від використання PPTP.

Порівняння з OpenVPN:

- **Безпека:** OpenVPN набагато безпечніший за PPTP.
- **Рекомендації:** Використання PPTP не рекомендується для критичних систем, тоді як OpenVPN є надійним рішенням.

4. L2TP/IPSec (Layer 2 Tunneling Protocol over IPSec)

Опис:

L2TP в поєднанні з IPSec забезпечує захищений тунель для передачі даних.

Переваги:

- **Безпека:** Поєднує функціональність L2TP та захист IPSec.
- **Підтримка на різних платформах:** Вбудована підтримка у багатьох операційних системах.

Недоліки:

- **Складність налаштування:** Потребує детального налаштування та може бути складним для недосвідчених користувачів.
- **Проблеми з брандмауерами та NAT:** Може мати труднощі при проходженні через мережеві обмеження.

Порівняння з OpenVPN:

- **Гнучкість:** OpenVPN легше проходить через NAT та брандмауери.

- **Простота використання:** OpenVPN може бути простішим у налаштуванні для багатьох сценаріїв.

5. SSTP (Secure Socket Tunneling Protocol)

Опис:

SSTP — протокол VPN, розроблений Microsoft, який використовує HTTPS для встановлення тунелю.

Переваги:

- **Проходження через брандмауери:** Використання порту 443 дозволяє легко проходити через більшість брандмауерів.
- **Вбудована підтримка в Windows:** Інтегрований у операційну систему Windows.

Недоліки:

- **Обмежена підтримка на інших платформах:** Менше доступний на не-Windows системах.
- **Закритий протокол:** Відсутність відкритого коду може бути проблемою для деяких організацій.

Порівняння з OpenVPN:

- **Крос-платформеність:** OpenVPN підтримує більше платформ.
- **Відкритість:** OpenVPN є відкритим та прозорим, що дозволяє перевіряти та модифікувати код.

1.5 Порівняння з аналогічними проєктами

На сучасному етапі розвитку кібербезпеки важливе значення має розробка систем, які забезпечують інтеграцію недорогого апаратного забезпечення з програмними інструментами для досягнення високого рівня захищеності мереж і систем. Особливої популярності набули рішення на базі платформ Raspberry Pi, які відзначаються доступністю, універсальністю, а також можливістю адаптації для потреб моніторингу, управління мережами та реагування на загрози. Додатково, інтеграція з Telegram-ботами для віддаленого управління стає все більш поширеною завдяки її ефективності, зручності та здатності забезпечити оперативне реагування на інциденти.

Далі буде проведено порівняння декількох проєктів, які мають схожість з розробкою, представленою в межах магістерської роботи. Порівняння здійснено з метою виявлення сильних і слабких сторін існуючих рішень та визначення унікальних можливостей запропонованого підходу.

1. Проєкт Pwnagotchi

Опис проєкту:

Pwnagotchi — це автономний пристрій на базі Raspberry Pi Zero W, який використовує машинне навчання для оптимізації захоплення Wi-Fi handshake-пакетів з метою подальшого аналізу безпеки бездротових мереж [7]. Пристрій працює у пасивному режимі, навчаючись на основі оточення та вдосконалюючи свої методи збору даних.

Схожість з магістерською роботою:

- **Використання Raspberry Pi** як основної апаратної платформи.
- **Фокус на безпеці мережі**, пов'язаний із завданнями кібербезпеки.

Відмінності:

- **Цільове призначення:** Pwnagotchi спеціалізується на бездротових мережах, тоді як магістерська робота може охоплювати ширший спектр задач.
- **Відсутність інтеграції з Telegram-ботом.**

2. Проєкт Pi-hole

Опис проєкту:

Pi-hole — це мережевий блокувальник реклами та трекерів, який працює на Raspberry Pi. Він діє як DNS-сервер, фільтруючи запити до небажаних доменів та підвищуючи безпеку та конфіденційність користувачів мережі [8].

Схожість з магістерською роботою:

- **Використання Raspberry Pi** для забезпечення безпеки мережі.
- **Підвищення рівня безпеки мережі** через блокування небажаних запитів.

Відмінності:

- **Фокус на блокуванні реклами** замість управління мережею та моніторингу кіберзагроз.
- **Відсутність Telegram-інтеграції** для сповіщень або управління.

3. Проєкт RaspiWiFi

Опис проєкту:

RaspiWiFi — це проєкт, який дозволяє швидко налаштовувати бездротові точки доступу на Raspberry Pi та забезпечувати їхнє управління через веб-інтерфейс. Він спрощує процес налаштування мережі та надає зручний спосіб керування підключеннями [9].

Схожість з магістерською роботою:

- **Використання Raspberry Pi** для завдань мережевого управління.
- **Мережеве управління:** RaspiWiFi надає інструменти для управління підключеннями.

Відмінності:

- **Інтерфейс управління через веб:** На відміну від Telegram-бота, що використовується в магістерській роботі.
- **Функціональність безпеки:** RaspiWiFi не спеціалізується на кібербезпеці чи моніторингу загроз.

4. Проєкт Telegram Bot With Raspberry Pi

Опис проєкту:

Цей проєкт демонструє, як створити Telegram-бота для управління Raspberry Pi, зокрема для виконання команд і контролю підключених пристроїв. Бот може взаємодіяти з користувачем через Telegram, надсилаючи статуси чи виконуючи запити.

Схожість з магістерською роботою:

- **Використання Telegram-бота** для виконання команд на Raspberry Pi.
- **Дистанційне управління пристроєм** через Telegram.

Відмінності:

- **Базовий функціонал для управління пристроями:** Проєкт не має повноцінних інструментів для кібербезпеки.
- **Відсутність інтеграції з OpenVPN та мережею безпеки,** як у магістерській роботі.

Критерій	**Магістерська робота**	**Pwnagotchi**	**Pi-hole**	**RaspiWiFi**	**Telegram Bot with Raspberry Pi**
Використання Raspberry Pi	✓	✓	✓	✓	✓
Telegram-бот для управління	✓	✕	✕	✕	✓
Фокус на кібербезпеці	✓	✓	✓	✕	✕
Інтеграція з OpenVPN	✓	✕	✕	✕	✕
Автоматизація та моніторинг	✓	✓	✓	✓	✓

Критерій	**Магістерська робота**	**Pwnagotchi**	**Pi-hole**	**RaspiWiFi**	**Telegram Bot with Raspberry Pi**
Управління мережею	✓	✕	Частково	✓	✕

Таблиця 1.1 – Порівняльний аналіз

Висновки з порівняння

Аналіз показує, що магістерська робота поєднує в собі кілька ключових компонентів:

- **Використання Raspberry Pi** як основної платформи для забезпечення мережевої безпеки та управління.
- **Інтеграція OpenVPN** для забезпечення безпечного віддаленого доступу.
- **Застосування Telegram-бота** для віддаленого управління та моніторингу.
- **Орієнтація на кібербезпеку,** включаючи моніторинг, управління та захист мережі.

Переваги магістерської роботи над аналогами

- **Комплексний підхід:** Поєднання апаратних та програмних компонентів для створення цілісного рішення.
- **Інтеграція Telegram-бота:** Забезпечує зручний спосіб управління та моніторингу системи з будь-якого місця.
- **Масштабованість та адаптивність:** Може бути адаптований під різні потреби й розширений додатковими функціями.

x

2 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ ДЛЯ БЕЗПЕЧНОГО ПЕНТЕСТУ З ВИКОРИСТАННЯМ RASPBERRY PI

2.1 Налаштування центрального сервера

2.1.1 Вибір хостингу та створення сервера на Kamatera

Для реалізації центрального сервера обрано хмарний сервіс Kamatera [10], який надає можливість гнучкого налаштування параметрів та забезпечує високу доступність інфраструктури. Операційна система для сервера — **Ubuntu 24.04 LTS** — вибрана через її стабільність, регулярні оновлення та тривалу підтримку, що є важливими факторами для безпечної роботи системи.

Процес створення віртуального сервера:

Після переходу до панелі керування Kamatera можна почати створення сервера, обравши параметри, які забезпечать оптимальні характеристики для пентест-системи.

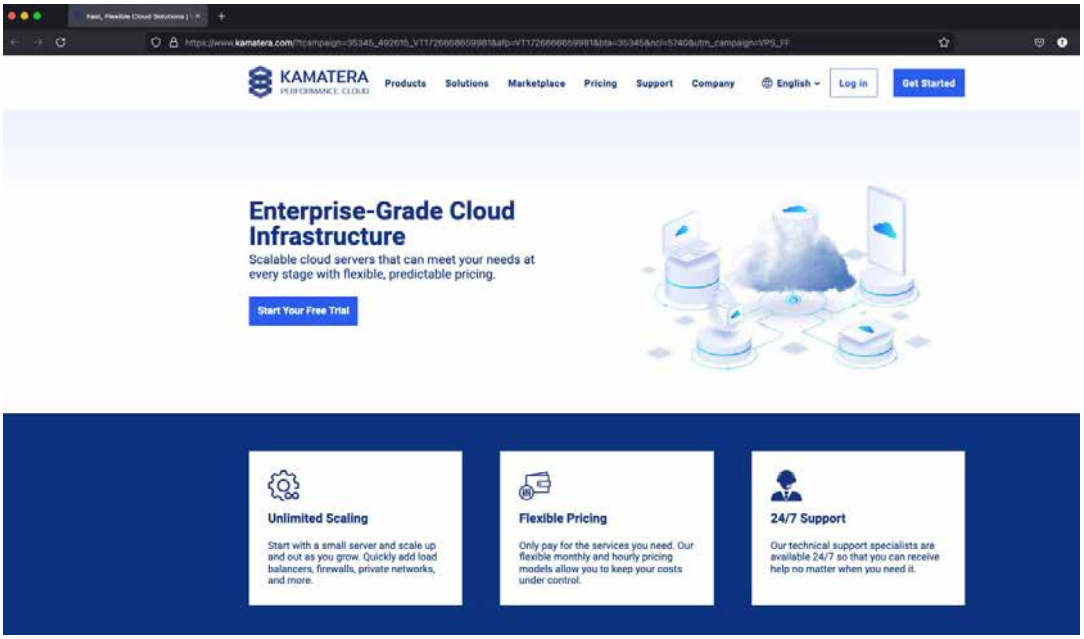


Рисунок 2.1 – Головна сторінка VPS платформи Kamatera

1. **Оберіть опцію створення нового сервера — "Create New Server"** у панелі керування.
2. **Налаштування параметрів сервера**

Для ефективної роботи центрального сервера під час виконання завдань пентесту необхідно налаштувати його параметри, відповідно до специфіки проєкту. Kamatera пропонує інтуїтивний процес налаштування сервера, який включає кілька кроків, детально описаних нижче.

1. Вибір зони розміщення (Choose Zone)

- **Data Center Location** — для мінімізації мережевих затримок і покращення продуктивності з'єднань обирається дата-центр, який географічно найближчий до кінцевих користувачів. Це дозволяє зменшити час відгуку сервера та забезпечити стабільність його роботи.

2. Вибір образу системи (Choose an Image)

- **Операційна система (Operating System)** — для сервера обирається **Ubuntu Server 24.04 LTS**, яка є надійною платформою для розгортання служб пентесту. Ця операційна система відзначається високою стабільністю, підтримкою сучасних технологій безпеки та довгостроковою підтримкою оновлень.

3. Вибір характеристик сервера (Choose Server Specs)

- **Тип сервера (TYPE)** — Kamatera пропонує чотири варіанти типів серверів, кожен із яких призначений для різних завдань:
    - **B (General Purpose)** — сервер загального призначення з гарантованими ресурсами CPU для стабільної продуктивності, що підходить для більшості стандартних задач. Це оптимальний варіант для завдань пентесту.
    - **D (Dedicated)** — сервери з виділеними фізичними ядрами CPU (2 потоки) та гарантованими ресурсами, рекомендовані для критично важливих додатків із високими вимогами до продуктивності.
    - **T (Burstable)** — сервери з можливістю тимчасового збільшення обчислювальної потужності. При перевищенні середнього використання CPU понад 10% можуть застосовуватись додаткові збори, що підходить для задач із перемінним навантаженням.

- **A (Availability)** — сервери без гарантованих ресурсів, придатні для задач, де потрібна висока доступність, але не критична продуктивність.

Для даного проєкту обрано **Type B (General Purpose)**, оскільки він забезпечує необхідну продуктивність та стабільність без додаткових витрат на виділені ресурси.

- **Обчислювальні ресурси (CPU та RAM)** — для початкової конфігурації обрано один віртуальний процесор (1 vCPU) та 1 GB оперативної пам’яті (RAM). Це мінімальні параметри, які відповідають вимогам проєкту. У разі необхідності ресурси можуть бути збільшені для підвищення продуктивності.
- **SSD Disk** — використовується SSD-накопичувач об’ємом 20 GB, що забезпечує швидкий доступ до даних і достатній обсяг пам’яті для початкових налаштувань системи. У разі потреби обсяг пам'яті може бути збільшений для зберігання додаткових даних і результатів тестів.

4. Налаштування мережі (Choose Networking)

- **Public Internet Network** — для забезпечення доступу до сервера з Інтернету активується параметр **Public Internet Network**. Це дозволяє серверу приймати підключення ззовні та взаємодіяти з іншими компонентами системи, такими як клієнтські пристрої та боти.
- **Private Local Network** — параметр приватної локальної мережі може бути використаний для додаткового захисту, однак для поточного проєкту цей параметр є необов'язковим.

5. Додаткові налаштування (Advanced Configuration / Finalize Settings)

- **Password** — для користувача root встановлюється надійний пароль. Це є важливим аспектом для забезпечення безпечного доступу до сервера.
- **Validate Password** — повторне введення пароля для підтвердження його правильності.
- **Power On Servers** — сервер автоматично активується після завершення налаштувань, що дозволяє негайно перейти до подальшого налаштування та встановлення необхідних служб.

6. Цикл оплати та ціноутворення (Billing Cycle & Pricing)

- Kamatera пропонує два варіанти циклів оплати:
    - **Monthly Billing Cycle** — місячний цикл оплати, який підходить для тривалих проєктів, оскільки забезпечує стабільні та прогнозовані витрати.
    - **Hourly Billing Cycle** — погодинний цикл оплати, зручний для короткострокових тестових задач, що дозволяє оплачувати тільки фактичний час використання сервера.

3. Після введення налаштувань та перевірки їх відповідності вимогам натисніть **"Create Server"**, щоб ініціювати процес розгортання віртуального сервера.

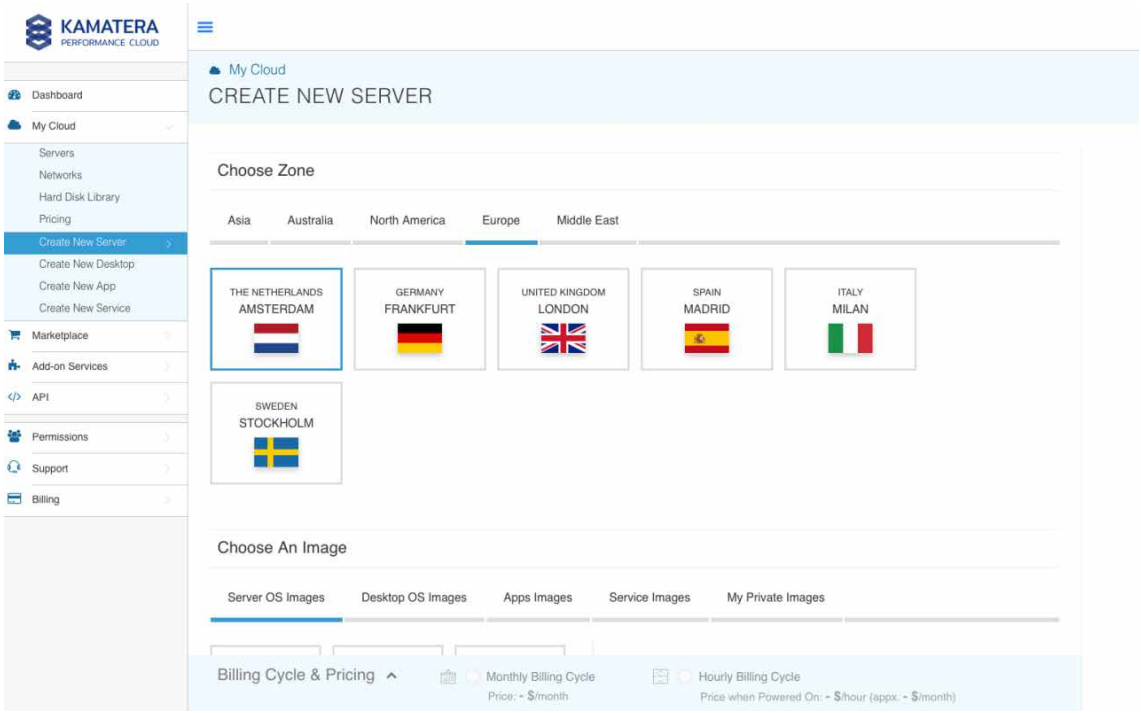


Рисунок 2.2 – Вибір параметрів і створення сервера

Рисунок 2.3 – Перегляд статусу розгортання VPS

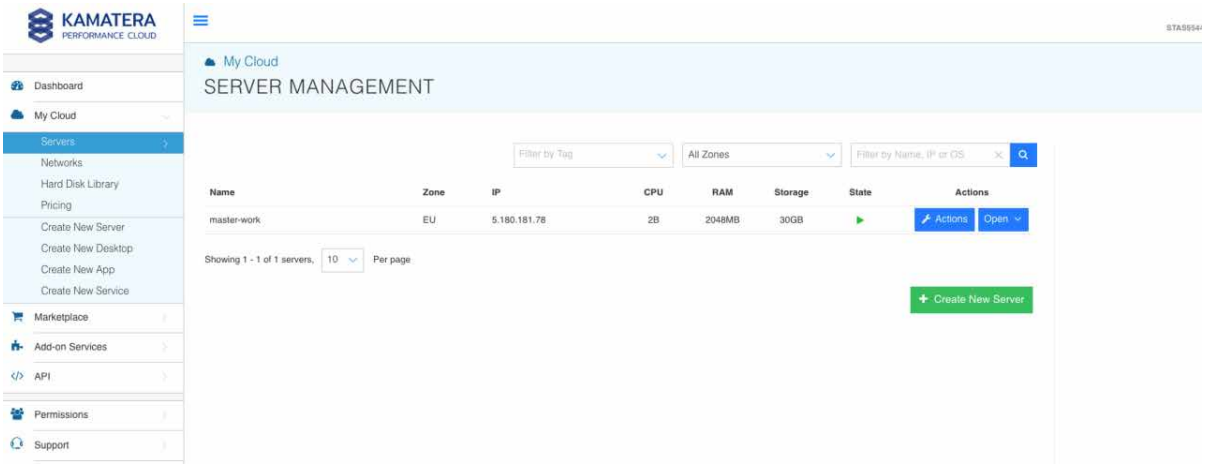


Рисунок 2.4 – Перегляд статусу розгортання VPS

2.1.2 Підключення до сервера та оновлення системи

Після створення сервера необхідно здійснити підключення до нього через SSH для проведення початкових налаштувань і оновлення системи.

Підключення через SSH

1. Запуск терміналу

На локальному комп'ютері відкривається термінал для віддаленого підключення до сервера.

2. Команда для підключення

У терміналі вводиться команда:

```
ssh root@<IP-адреса сервера>
```

де <IP-адреса сервера> — це публічна IP-адреса сервера, надана Kamatera.

3. Підтвердження автентичності

При першому підключенні потрібно підтвердити автентичність хоста, ввівши "yes".

4. Введення пароля root

Вводиться пароль адміністратора, встановлений під час створення сервера.

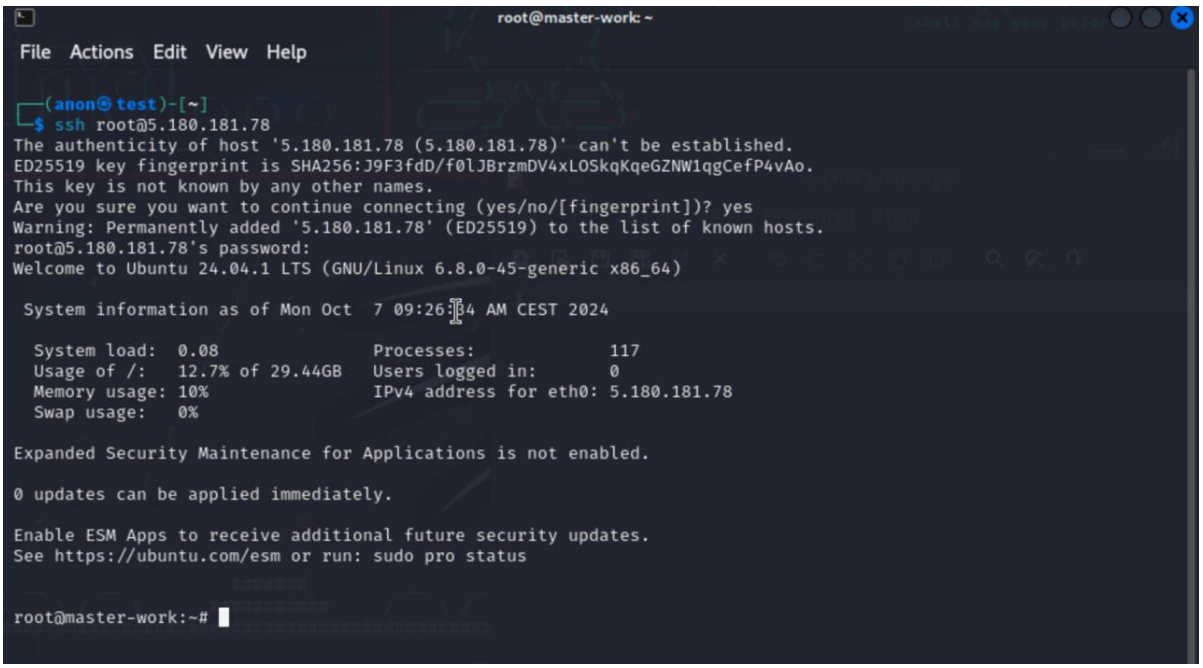


Рисунок 2.5 – Підключення до VPS

Оновлення системи

Оновлення списку пакетів та встановлення оновлень

Для оновлення системи виконується команда:

```
sudo apt update && sudo apt upgrade -y
```

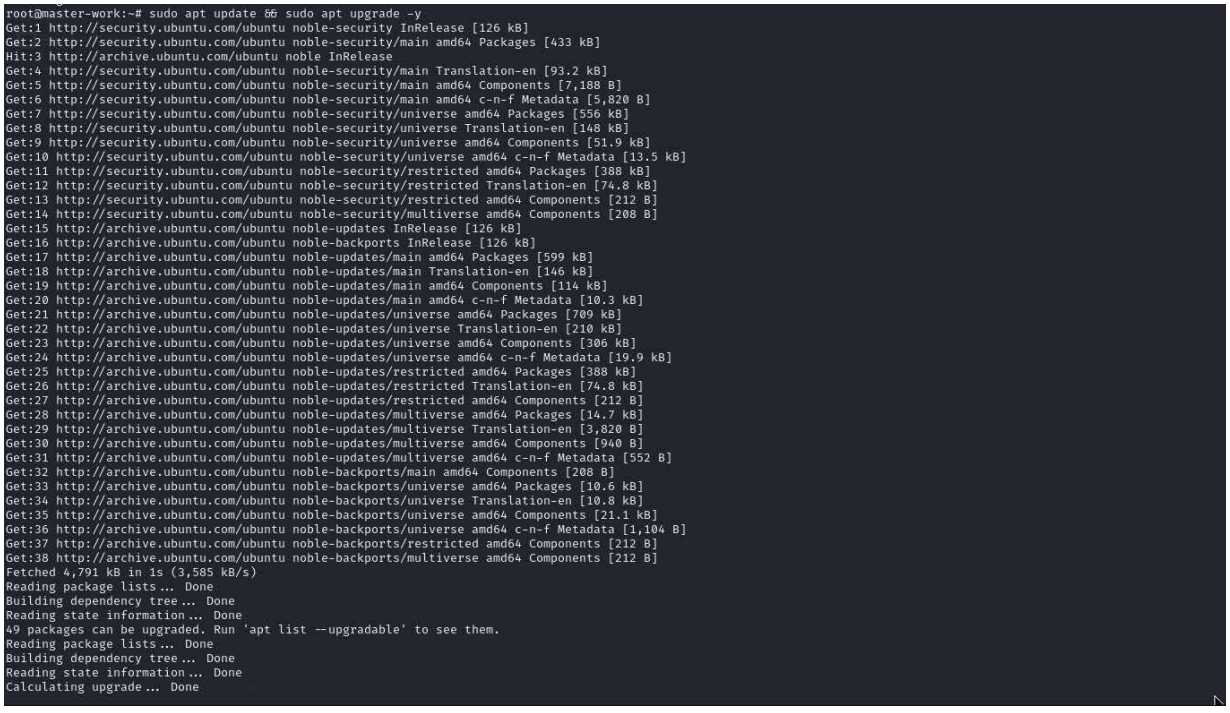


Рисунок 2.6 – Оновлення та встановлення списку пакетів на VPS

2.1.3 Встановлення необхідних пакетів

Для коректної роботи системи необхідно встановити ряд пакетів, таких як OpenVPN для створення VPN-з'єднання, Easy-RSA для керування сертифікатами та Python для розробки Telegram-бота.

1. Встановлення OpenVPN та Easy-RSA:

```
sudo apt install openvpn easy-rsa -y
```

openvpn — програмне забезпечення для створення захищених віртуальних приватних мереж.

easy-rsa — утиліта для створення сертифікатів та ключів для OpenVPN.

2. Встановлення Python 3 та пов'язані пакети:

```
sudo apt install python3 python3-venv python3-pip -y
```

python3 — інтерпретатор мови Python версії 3.

python3-venv — модуль для створення віртуальних середовищ Python.

python3-pip — пакетний менеджер для встановлення Python-бібліотек.

Рисунок 2.7 – Встановлення пов’язаних пакетів на VPS

2.2 Налаштування OpenVPN на центральному сервері

OpenVPN забезпечує захищене з'єднання між сервером і клієнтами через шифрований тунель, що дозволяє безпечно передавати дані та керувати пристроями у віддалених мережах.

2.2.1 Клонування репозиторію OpenVPN-інсталятора

Для спрощення процесу встановлення та налаштування OpenVPN використовується скрипт з репозиторію Angristan [11], який автоматизує більшість етапів конфігурації.

1. Клонування репозиторію:

```
git clone https://github.com/angristan/openvpn-install.git
```

git clone — команда для клонування репозиторію з GitHub, яка копіює всі файли з репозиторію до локального каталогу.

Рисунок 2.8 – Клонування репозиторію

2. Перенесення скрипту встановлення:

```
mv openvpn-install/openvpn-install.sh ./
```

mv — команда для переміщення файлів. Ця команда переміщує скрипт openvpn-install.sh до поточної директорії для зручності.

3. Надання прав на виконання скрипту:

```
chmod +x openvpn-install.sh
```

chmod +x — надає права на виконання файлу. Це необхідно для того, щоб скрипт можна було запустити.

2.2.2 Запуск скрипту встановлення OpenVPN

1. Запуск скрипту:

```
sudo ./openvpn-install.sh
```

sudo — виконує команду з правами адміністратора.

./openvpn-install.sh — запускає скрипт встановлення, який запитає декілька налаштувань для повного встановлення сервера OpenVPN.

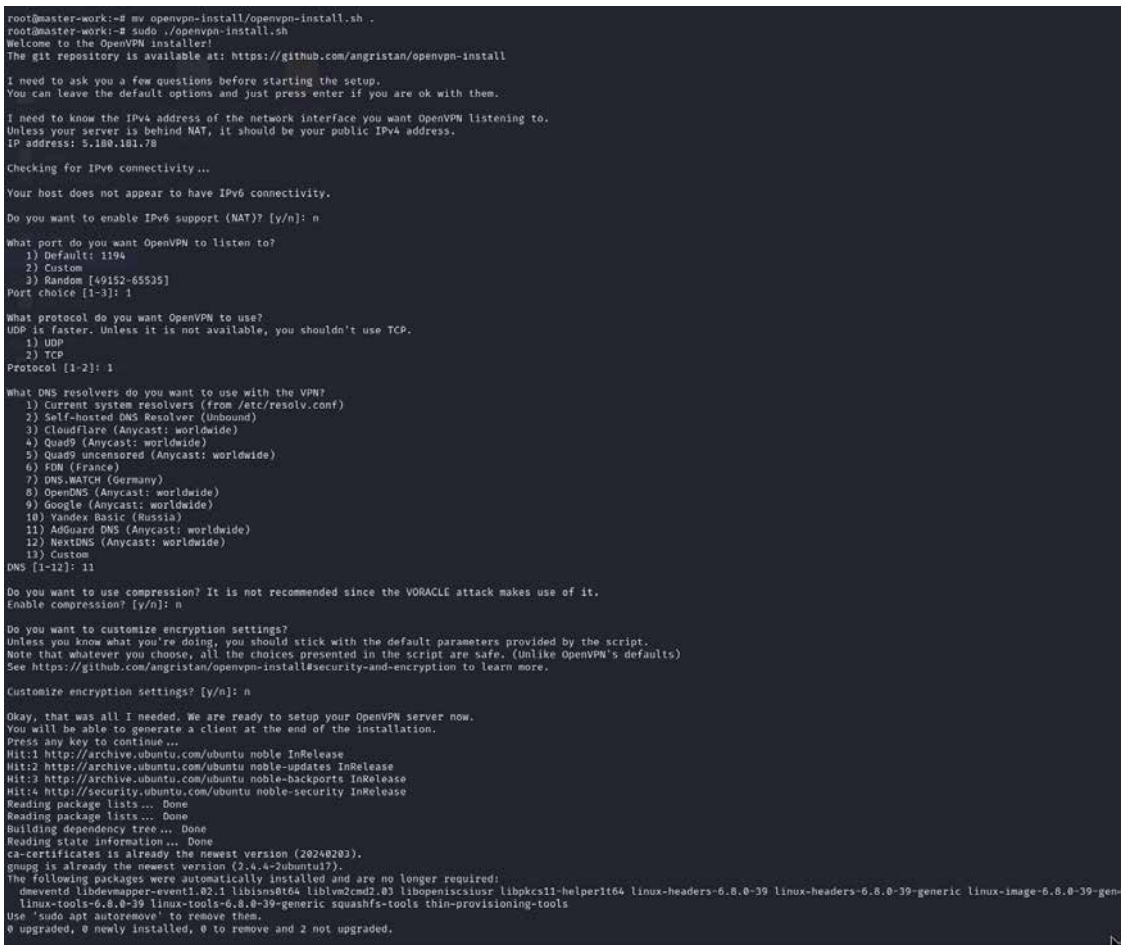


Рисунок 2.9 – Перенесення та встановлення скрипту для встановлення OpenVPN

2. Відповіді на запитання скрипту:

Під час виконання скрипт запитає кілька параметрів для налаштування:

- **IP-адреса сервера:** Скрипт автоматично визначить IP-адресу сервера, але її можна змінити за потреби.
- **Протокол:** Було обрано UDP для швидшого з'єднання, оскільки UDP краще підходить для VPN-трафіку.
- **Порт:** За замовчуванням встановлюється порт 1194. Його можна змінити, якщо потрібно.
- **DNS-сервери:** Можна обрати вбудовані DNS-сервери або відомі, наприклад, Google DNS.
- **Назва клієнта:** Введено raspberry для клієнта Raspberry Pi, який підключатиметься до цього VPN.

- **Шифрування та сертифікати:** Залишив параметри за замовчуванням або якщо потрібно, можна налаштувати відповідно до політики безпеки.

3. Завершення встановлення:

Після завершення встановлення скрипт згенерує клієнтський конфігураційний файл raspberry.ovpn, який необхідний для налаштування VPN-клієнта на Raspberry Pi.

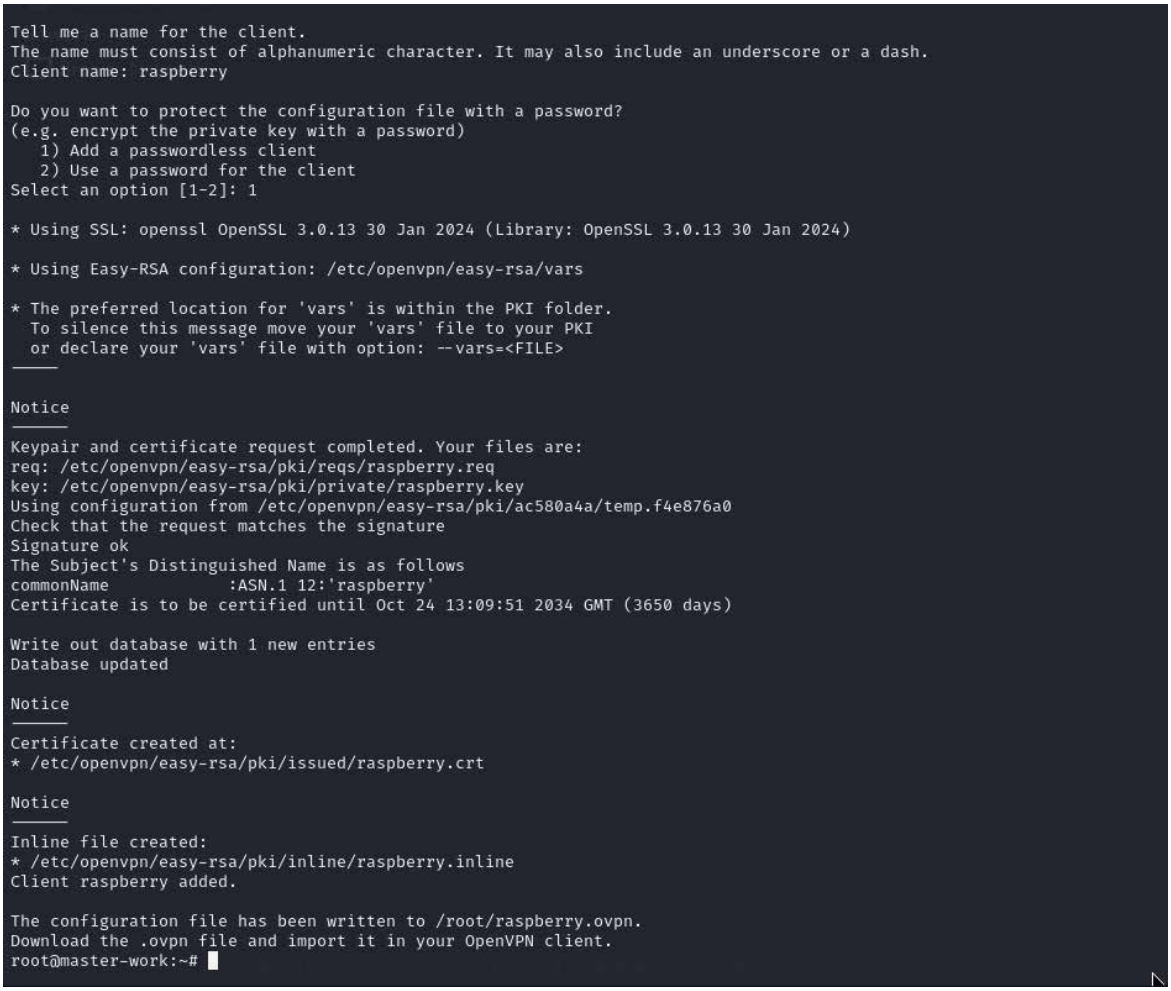


Рисунок 2.10 – Результат встановлення OpenVPN

2.2.3 Налаштування конфігураційних файлів OpenVPN

Після завершення встановлення необхідно внести додаткові налаштування в конфігураційні файли для забезпечення коректної роботи системи.

1. Редагування файлу server.conf:

```
sudo nano /etc/openvpn/server.conf
```

nano — текстовий редактор у терміналі. За допомогою цієї команди відкривається файл конфігурації OpenVPN.

2. Додавання додаткових налаштувань у server.conf:

```
client-config-dir ccd
client-to-client
```

client-config-dir ccd — вказує OpenVPN використовувати індивідуальні налаштування для клієнтів з директорії ccd.

client-to-client — дозволяє клієнтам у VPN-мережі спілкуватися один з одним, що може бути корисним для взаємодії між сервером і Raspberry Pi.

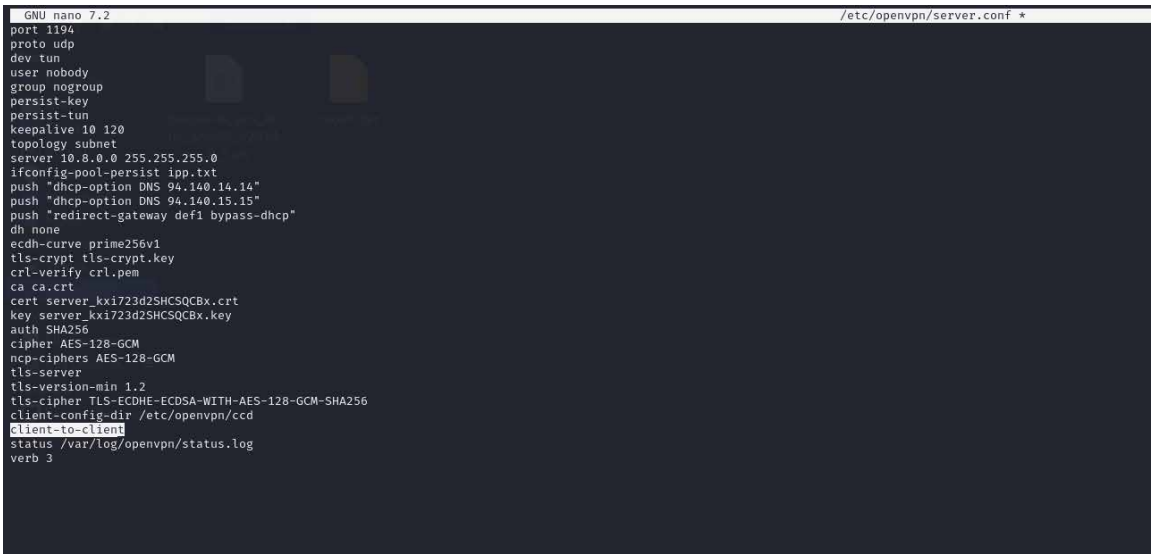


Рисунок 2.11 – Результат встановлення OpenVPN

3. Збереження та вихід з редактора:
   - Натисніть Ctrl + O, щоб зберегти файл.
   - Натисніть Enter, щоб підтвердити збереження.
   - Натисніть Ctrl + X для виходу з редактора.

4. Створення директорії ccd:

```
sudo mkdir /etc/openvpn/ccd
```

mkdir — команда для створення нової директорії. Директорія ccd буде містити індивідуальні конфігураційні файли для кожного VPN-клієнта.

5. Створення файлу конфігурації для клієнта Raspberry Pi:

```
sudo nano /etc/openvpn/ccd/raspberry
```

6. Додавання статичної IP-адреси для клієнта Raspberry Pi:

```
ifconfig-push 10.8.0.10 255.255.255.0
```

ifconfig-push — призначає статичну IP-адресу клієнту у VPN-мережі.

10.8.0.10— IP-адреса, яку отримає клієнт Raspberry Pi при підключенні.

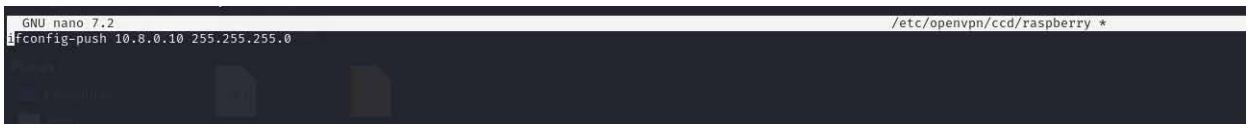


Рисунок 2.12 – Додавання статичної IP адреси

7. Збереження та вихід з редактора:
   - Натисніть Ctrl + O, щоб зберегти файл.
   - Натисніть Enter, щоб підтвердити збереження.
   - Натисніть Ctrl + X для виходу з редактора.

2.2.4 Перезапуск служби OpenVPN

Після внесення змін до конфігураційних файлів необхідно перезапустити службу OpenVPN, щоб застосувати нові налаштування.

1. Перезапуск служби OpenVPN:

```
sudo systemctl restart openvpn
```

systemctl restart openvpn — команда для перезапуску служби OpenVPN.

Рисунок 2.13 – Перезапуск служби OpenVPN

2. Перевірка статусу служби OpenVPN:

Щоб переконатися, що служба OpenVPN працює коректно і без помилок, потрібно виконати наступну команду:

```
sudo systemctl status openvpn
```

Якщо служба працює належним чином, на екрані відобразиться статус active (running).

Після виконання цих кроків OpenVPN налаштовано на сервері, і він готовий для підключення клієнтських пристроїв через захищений тунель.

2.3 Налаштування та перевірка підключення Raspberry Pi до сервера

Після налаштування OpenVPN на центральному сервері необхідно підготувати Raspberry Pi, щоб він міг підключатися до VPN, а також налаштувати автоматичний запуск VPN-клієнта. Це забезпечить захищене з'єднання між Raspberry Pi та центральним сервером.

2.3.1 Встановлення Kali Linux на Raspberry Pi за допомогою Raspberry Pi Imager

Для встановлення Kali Linux на Raspberry Pi було використано програмне забезпечення **Raspberry Pi Imager**, яке дозволяє швидко записати необхідний образ на SD-карту.

1. Було завантажено та встановлено Raspberry Pi Imager з офіційного сайту Raspberry Pi [12].

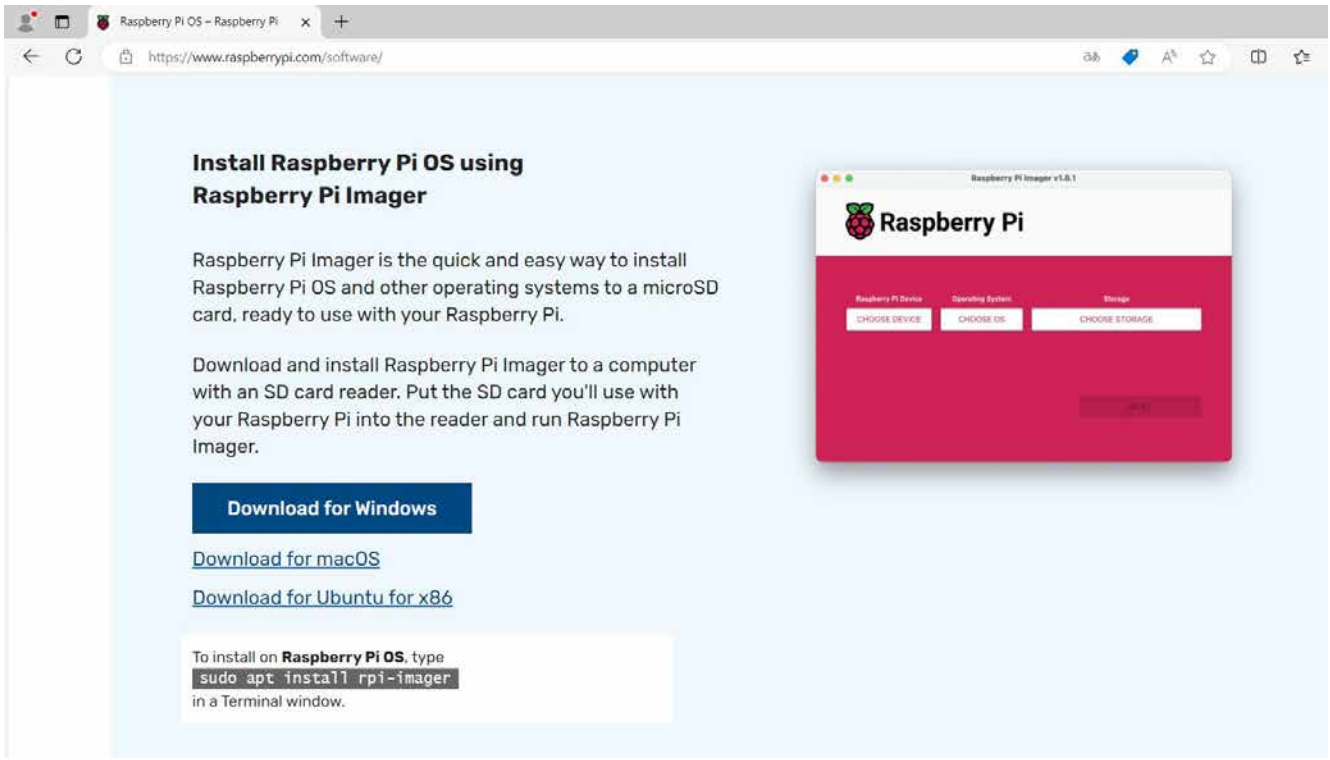


Рисунок 2.14 – Завантаження ПЗ Raspberry Pi Imager

Рисунок 2.15 – Встановлення ПЗ Raspberry Pi Imager

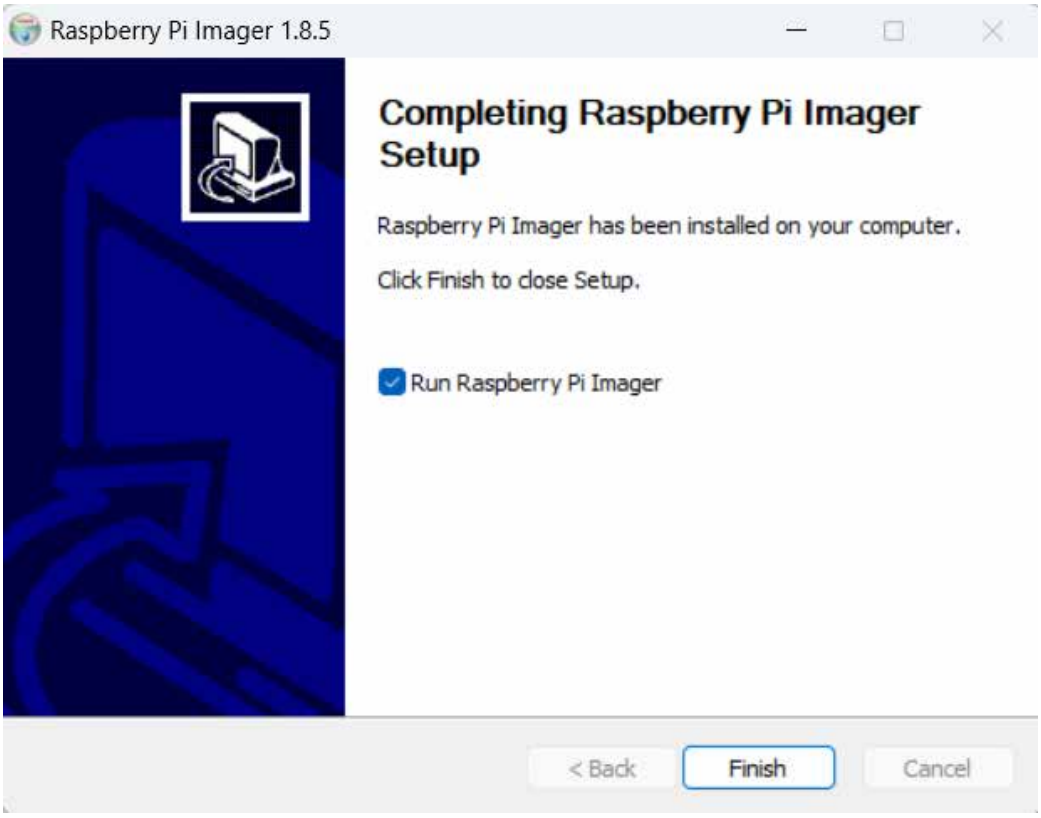


Рисунок 2.16 – Встановленний ПЗ Raspberry Pi Imager

2. Вставлено SD-карту у комп’ютер, після чого в **Raspberry Pi Imager** обрано операційну систему **Kali Linux** зі списку доступних ARM-образів.

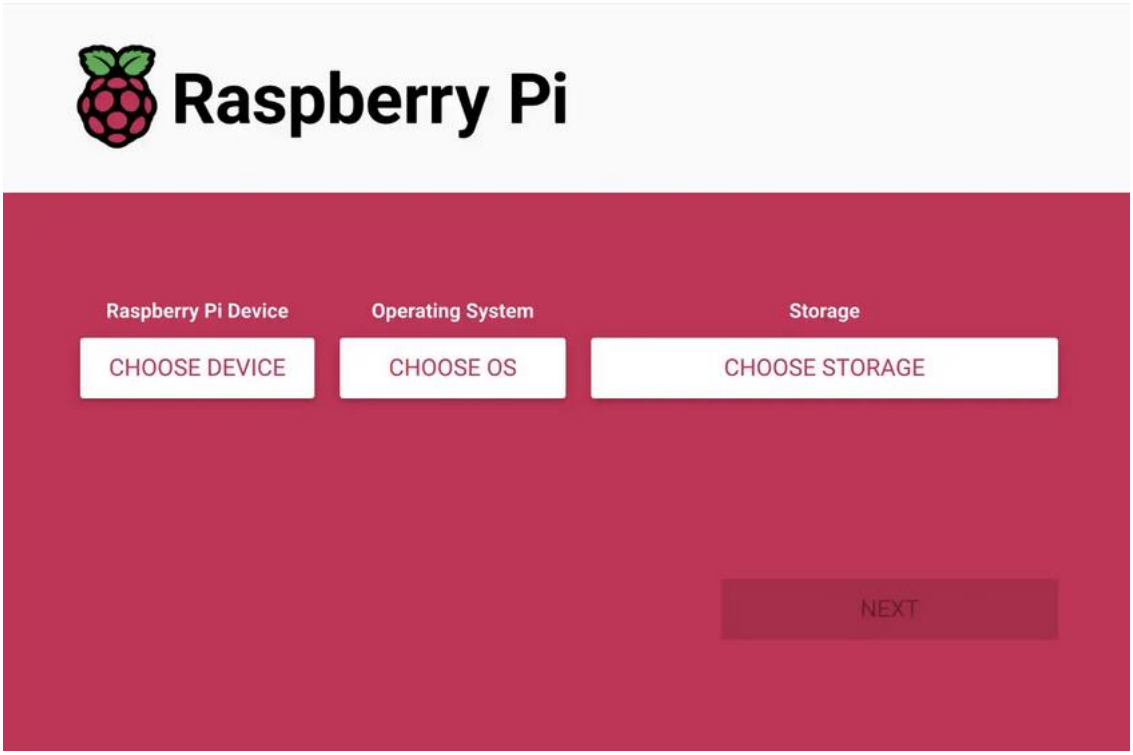


Рисунок 2.17 – Початковий екран ПЗ Raspberry Pi Imager

3. Обрано SD-карту як цільовий пристрій і виконано запис образу на карту, після чого SD-карта була готова до використання з Raspberry Pi.

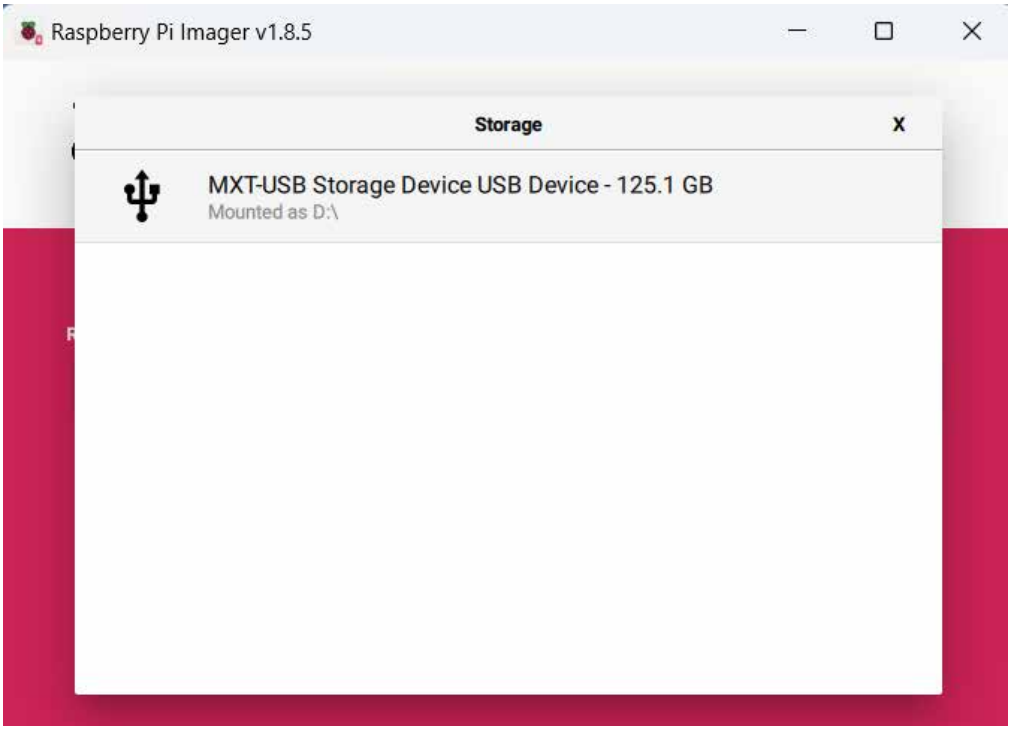


Рисунок 2.18 – Вибір SD карти

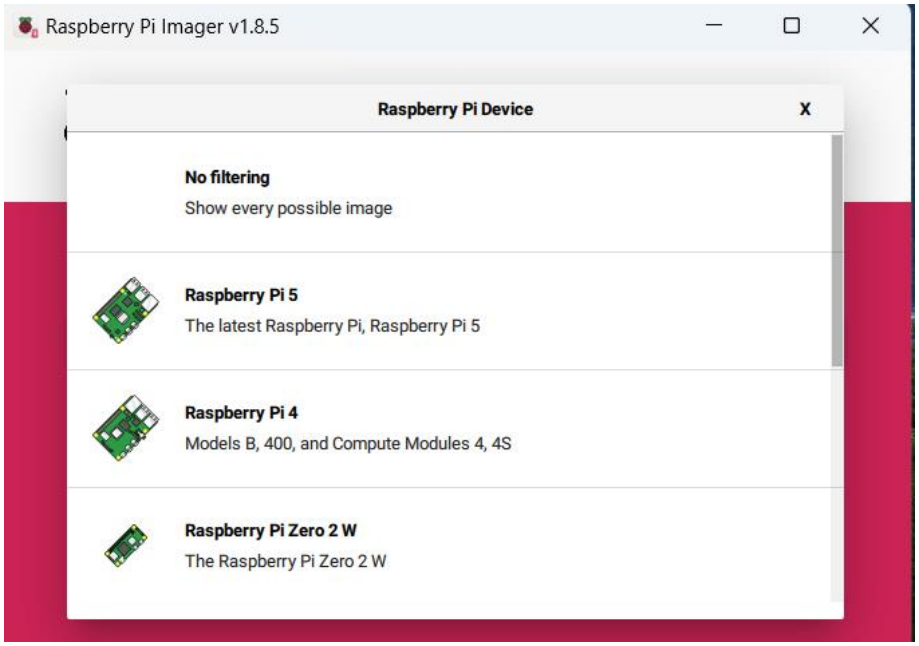


Рисунок 2.19 – Вибір моделі Raspberry

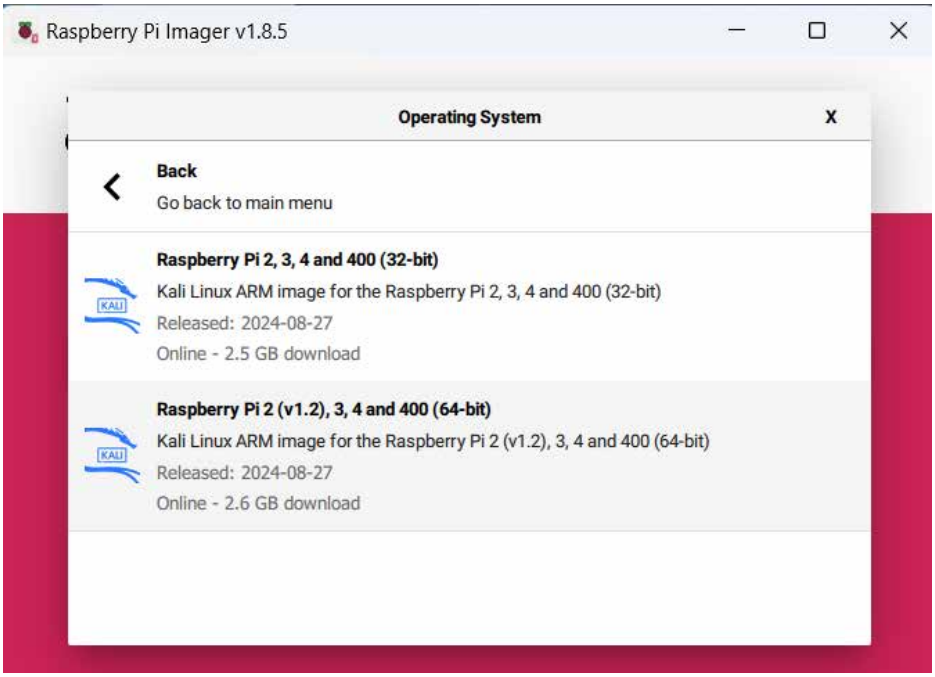


Рисунок 2.20 – Вибір системи

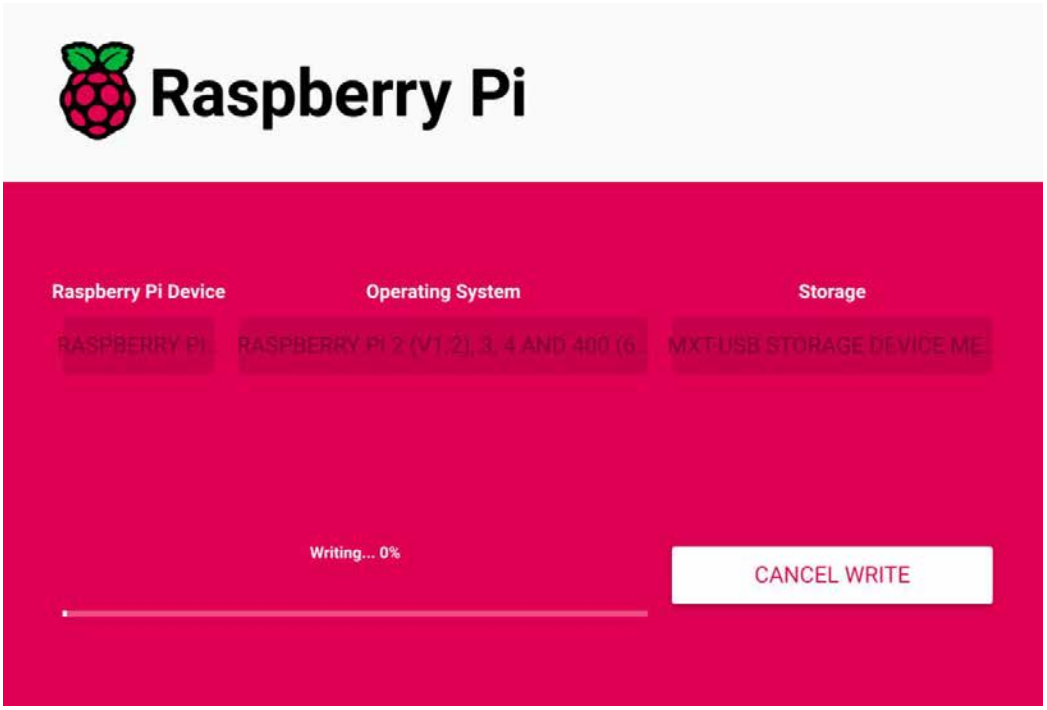


Рисунок 2.21 – Запис системи на SD-карту

Рисунок 2.22 – використання Raspberry Pi

4. Після завершення запису SD-карту вставлено у Raspberry Pi, і пристрій було увімкнено. Після завантаження системи виконано підключення до мережі (через Wi-Fi) та проведено оновлення системи для забезпечення стабільності й безпеки:

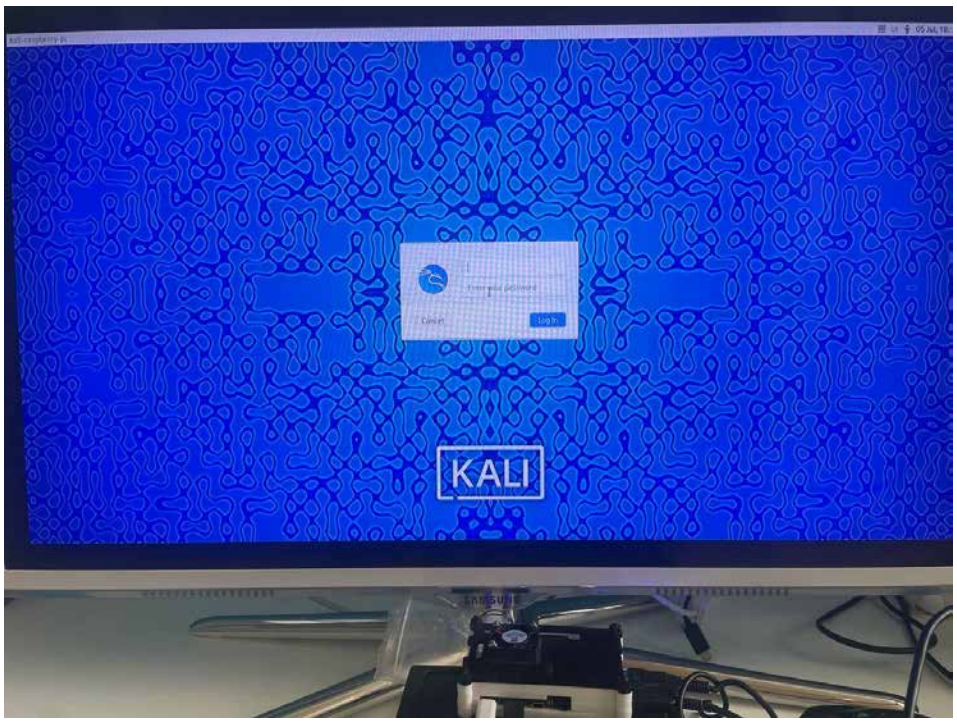


Рисунок 2.23 – Підключення до Raspberry Pi

Виконання команди:

```
sudo apt update && sudo apt upgrade -y
```

2.3.2 Налаштування OpenVPN на Raspberry Pi

Після встановлення Kali Linux було налаштовано VPN-клієнт на Raspberry Pi для підключення до центрального сервера.

1. Встановлено OpenVPN на Raspberry Pi:

```
sudo apt install openvpn -y
```

2. Клієнтський конфігураційний файл **raspberry.ovpn**, створений на центральному сервері, було скопійовано на Raspberry Pi за допомогою команди scp:

```
scp root@<IP-адреса сервера>:/root/raspberry.ovpn /home/kali/
```

<IP-адреса сервера> — публічна IP-адреса центрального сервера.

Рисунок 2.24 – Копіювання конфігураційного файлу

3. Конфігураційний файл було переміщено до директорії **/etc/openvpn/** та перейменовано на **raspberry.conf** для зручності:

```
sudo mv /home/kali/raspberry.ovpn /etc/openvpn/raspberry.conf
```

2.3.3 Налаштування автозапуску VPN-клієнта

Для забезпечення автоматичного підключення Raspberry Pi до VPN при кожному завантаженні системи було налаштовано автозапуск VPN-клієнта.

1. Активовано автозапуск служби OpenVPN при завантаженні системи:

```
sudo systemctl enable openvpn@raspberry
```

2. Службу OpenVPN було запущено:

```
sudo systemctl start openvpn@raspberry
```

3. Для перевірки статусу служби було виконано команду:

sudo systemctl status openvpn@raspberry

Якщо служба працює коректно, на екрані відображається статус **active (running)**, що свідчить про успішне підключення Raspberry Pi до центрального сервера через VPN.

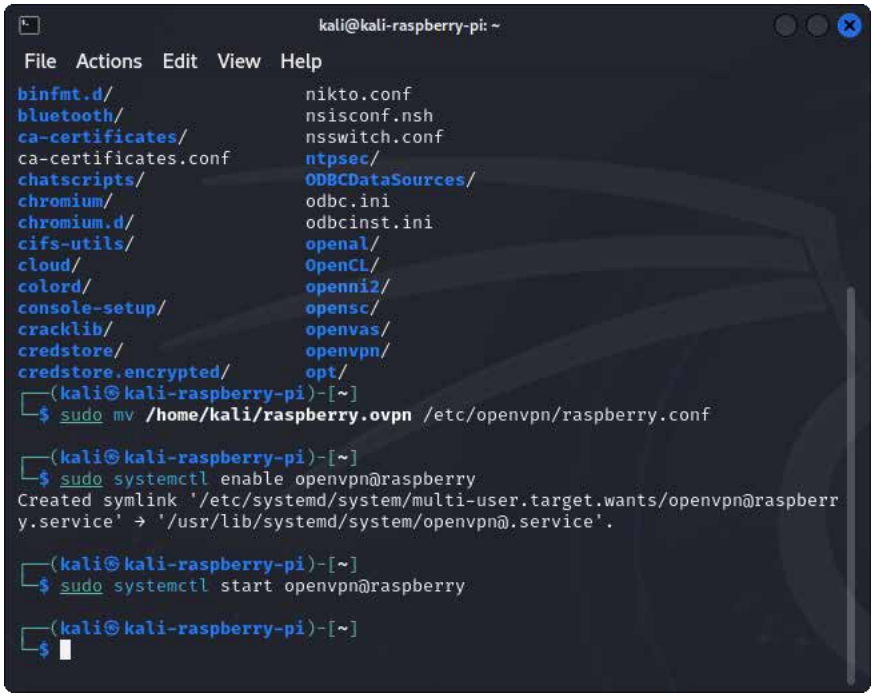


Рисунок 2.25 – Налаштування автозапуску OpenVPN

2.3.4 Перевірка стану VPN-з’єднання

Після налаштування VPN-клієнта було проведено перевірку з’єднання для підтвердження успішного підключення Raspberry Pi до VPN.

1. Для перевірки наявності інтерфейсу **tun0** було використано команду:

```
ifconfig tun0
```

Наявність інтерфейсу **tun0** з IP-адресою 10.8.0.10 свідчить про успішне підключення до VPN.

2. Для перевірки з'єднання з центральним сервером було здійснено пінг до його внутрішньої IP-адреси у VPN-мережі:

```
ping 10.8.0.1 -c 4
```

10.8.0.1 — IP-адреса центрального сервера у VPN-мережі.

-c 4 — параметр, що обмежує кількість відправлених пакетів до чотирьох для перевірки стабільності з’єднання.

2.4 Встановлення та налаштування інструментів для сканування на Raspberry Pi

У цьому розділі описано процес налаштування інструментів для сканування мережі на Raspberry Pi, зокрема OpenVAS, а також налаштування автоматичного сканування мережі за допомогою **Nmap**. Це дозволить автоматизувати процеси виявлення активних хостів та пошуку вразливостей у мережі.

2.4.1 Налаштування OpenVAS

1. Встановлення OpenVAS

OpenVAS надає можливість проводити детальний аналіз стану мережі та виявляти потенційні вразливості.

```
sudo apt install openvas -y
```

Рисунок 2.26 – Встановлення OpenVAS

2. Початкове налаштування OpenVAS

Після встановлення OpenVAS виконується початкове налаштування, яке підготовлює бази даних вразливостей і конфігурацію системи для подальших сканувань:

```
sudo gvm-setup
```

Ця команда запускає процес налаштування та оновлення баз даних вразливостей. Процедура може зайняти деякий час, оскільки завантажуються та індексуються великі обсяги інформації.

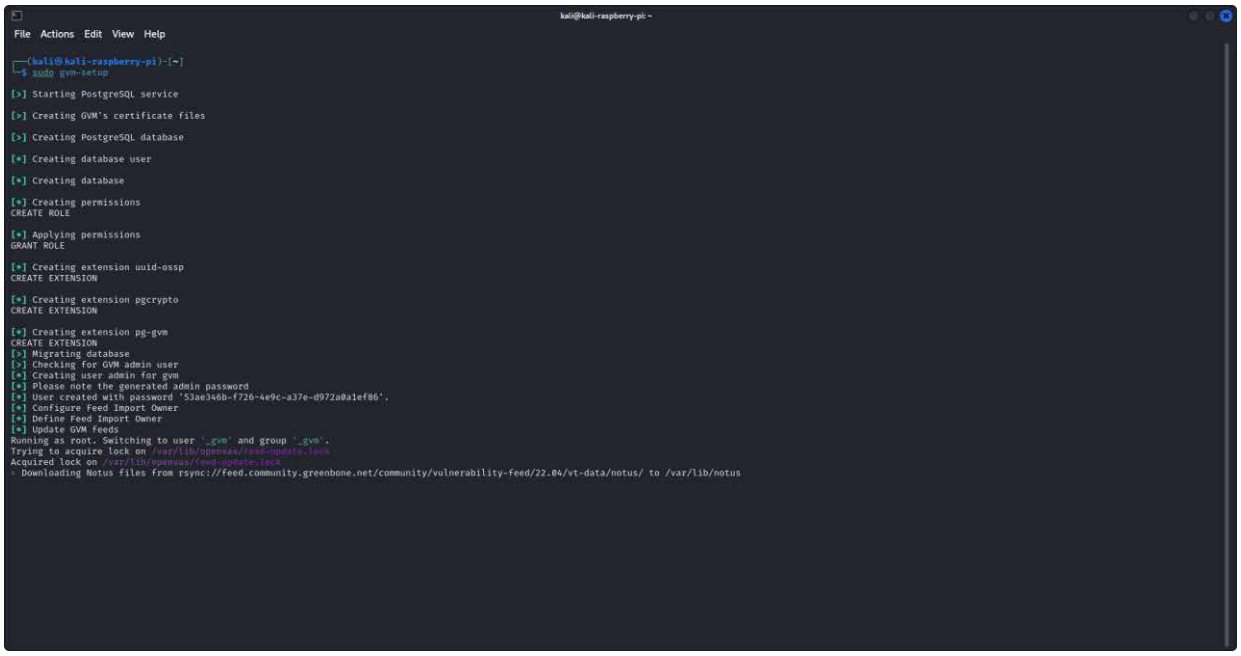


Рисунок 2.27 – Налаштування OpenVAS

3. Створення адміністратора та встановлення пароля

Для управління OpenVAS змінюється обліковий запис адміністратора:

```
sudo gvmd --user=admin --new-password=your_password
```

your_password — пароль, встановлений для користувача **admin**.

4. Перевірка встановлення

Для перевірки коректності налаштувань OpenVAS використовується команда:

```
sudo gvm-check-setup
```

5. Запуск OpenVAS

Після перевірки конфігурації служба OpenVAS запускається командою:

```
sudo gvm-start
```

Рисунок 2.28 – Запуск OpenVAS

6. Доступ до веб-інтерфейсу OpenVAS

Для управління OpenVAS використовується веб-інтерфейс. Доступ до нього здійснюється через браузер на Raspberry Pi:

```
https://localhost:9392
```

Для входу в систему використовується ім’я користувача **admin** та встановлений пароль.

2.4.2 Налаштування автоматичного сканування мережі на Raspberry Pi

Для регулярного сканування мережі створюється Bash-скрипт, який автоматично визначає мережевий діапазон для сканування, виконує пошук активних хостів за допомогою Nmap та зберігає результати у файл. Потім цей скрипт налаштовується для автоматичного запуску щогодини за допомогою cron-завдань.

1. Створення директорії для збереження результатів сканування

У домашній директорії створюється папка **scan_results** для зберігання результатів сканувань:

```
mkdir /home/kali/scan_results
```

2. Створення скрипту для автоматичного сканування мережі

Для виконання регулярного сканування мережі створюється скрипт, що автоматично визначає поточний мережевий інтерфейс і IP-діапазон, виконує сканування активних хостів та зберігає результати у файл.

1. Створення файлу скрипту **network_scan.sh**:

```
nano /home/kali/network_scan.sh
```

2. Вставлення наступного коду у файл:

```
#!/bin/bash
# Визначення мережі автоматично
INTERFACE=$(ip route | grep default | awk '{print $5}')
NETWORK_RANGE=$(ip -o -f inet addr show $INTERFACE | awk
'{print $4}')

# Перевірка чи вдалося визначити мережу
if [ -z "$NETWORK_RANGE" ]; then
```

```
        echo "Не вдалося визначити мережу" >&2
        exit 1
    fi

    # Виконуємо сканування nmap
    nmap -sn $NETWORK_RANGE >
/home/kali/scan_results/scan_results.txt
    echo "Сканування завершено $(date)" >>
/home/kali/scan_results/scan_results.txt
```

#!/bin/bash — вказує на використання Bash як інтерпретатора.

INTERFACE — визначає мережевий інтерфейс для інтернет-з'єднання.

NETWORK_RANGE — визначає IP-діапазон для сканування.

nmap -sn $NETWORK_RANGE — виконує "ping scan" для виявлення активних хостів у мережевому діапазоні.

Результати сканування зберігаються у файл **/home/kali/scan_results/scan_results.txt**, а в кінці додається час завершення сканування.

3. Збереження та вихід з редактора:
   - Натисніть Ctrl + O для збереження.
   - Натисніть Enter для підтвердження.
   - Натисніть Ctrl + X для виходу.

3. Надання прав на виконання скрипту

Для того щоб дозволити виконання скрипту, надаються відповідні права:

```
chmod +x /home/kali/network_scan.sh
```

4. Налаштування crontab для автоматичного запуску сканування щогодини

Щоб скрипт виконувався автоматично щогодини, налаштовується cron-завдання.

1. Відкриття редактора crontab:

```
crontab -e
```

2. Додавання наступного рядка до crontab для щогодинного запуску скрипту:

```
0 * * * * /home/kali/network_scan.sh
```

0 * * * * — cron-вираз, що відповідає запуску щогодини на початку кожної години.

home/kali/network_scan.sh — шлях до скрипту, який запускається cron-завданням.

3. Збереження та вихід з редактора crontab.

Після виконання цих кроків Raspberry Pi автоматично скануватиме мережу щогодини і зберігатиме результати в директорії **/home/kali/scan_results**. Це дозволяє централізовано отримувати інформацію про стан мережі та виявляти потенційні вразливості, що є важливим компонентом для подальшого аналізу безпеки мережі.

2.5 Розробка, налаштування та використання Telegram-бота

У цьому розділі буде розглянуто створення Telegram-бота, який забезпечує віддалене керування VPN-системою та інструментами сканування, такими як Nmap і OpenVAS. Telegram-бот дозволяє виконувати завдання, зокрема додавання та видалення VPN-клієнтів, запуск сканування мережі, завантаження результатів сканувань і моніторинг VPN-сесій. Завдяки цьому боту забезпечується зручний інтерфейс для безпечного пентеста.

2.5.1 Створення Telegram-бота

Для створення Telegram-бота використовується сервіс **BotFather** у Telegram, який дозволяє реєструвати нових ботів і надає токен доступу для ідентифікації бота в Telegram API.

1. Реєстрація бота через BotFather

- У Telegram шукаємо @BotFather.
- Потрібно відправити команду /newbot для створення нового бота.
- Також ввести обов’язково потребує ім'я для бота (наприклад, NulesMasterWorkBot) та унікальне ім'я користувача, яке має закінчуватися на "bot" (наприклад, nulesmasterwork_bot).
- Після завершення реєстрації BotFather надасть токен доступу, який використовується у коді для налаштування бота.

2.5.2 Налаштування Python-середовища для Telegram-бота

Для роботи Telegram-бота необхідно налаштувати Python-середовище на центральному сервері. Це дозволяє ізолювати залежності бота від системних бібліотек і забезпечити стабільність його роботи.

1. Створення директорії та віртуального середовища Python

```
mkdir ~/vpn_bot
cd ~/vpn_bot
python3 -m venv venv
source venv/bin/activate
```

2. Встановлення необхідних бібліотек

В моєму Telegram-бот використовує кілька бібліотек Python:

python-telegram-bot — для роботи з Telegram API.

paramiko — для SSH-з'єднань з Raspberry Pi.

scp — для передачі файлів через SSH.

gvm-tools — для взаємодії з OpenVAS.

```
pip install python-telegram-bot==20.0 paramiko scp gvm-tools
```

2.5.3 Основні функції Telegram-бота

Telegram-бот реалізований у файлі **vpn_bot.py**. Він надає користувачу можливість обирати різні дії через кнопкове меню, серед яких додавання та видалення VPN-клієнтів, запуск сканувань, завантаження результатів та моніторинг VPN-з'єднання. Розглянемо всі основні функції бота.

1. Головне меню

Головне меню Telegram-бота надає користувачу доступ до всіх основних функцій через інтерактивні кнопки.

```
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    if update.effective_user.id != AUTHORIZED_USER_ID:
        await update.effective_message.reply_text("✖ У вас немає доступу.")
        return ConversationHandler.END
    keyboard = [
        [InlineKeyboardButton("✚ Додати клієнта", callback_data="add_client")],
        [InlineKeyboardButton("━ Видалити клієнта", callback_data="remove_client")],
        [InlineKeyboardButton(" Активні клієнти", callback_data="list_clients")],
        [InlineKeyboardButton(" Переглянути файли сканування", callback_data="list_scans")],
        [InlineKeyboardButton(" Запустити OpenVAS сканування", callback_data="start_openvas_scan")],
        [InlineKeyboardButton(" Запустити Nmap сканування", callback_data="start_nmap_scan")],
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.effective_message.reply_text(
        " Вітаю! Виберіть дію з меню нижче:", reply_markup=reply_markup
    )
```

```
    return ConversationHandler.END
```

2. Додавання клієнта VPN

Функція `add_client_input` забезпечує додавання нового клієнта до VPN-системи. Вона приймає ім'я нового клієнта, викликає зовнішній скрипт **add_user.sh** для додавання, створює файл конфігурації **.ovpn** і надсилає його користувачу через Telegram.

```
async def add_client_input(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    client_name = update.message.text
    ovpn_file = f"{OVPN_OUTPUT_DIR}/{client_name}.ovpn"


    try:
        cmd_create = f"sudo bash {ADD_USER_SCRIPT} {client_name}"
        subprocess.run(cmd_create, shell=True, check=True)


        if os.path.exists(ovpn_file):
            with open(ovpn_file, "rb") as f:
                await update.message.reply_document(f,
filename=f"{client_name}.ovpn")
            os.remove(ovpn_file)
            await update.message.reply_text(
                f"✅ Клієнт <b>{escape(client_name)}</b> успішно
доданий, файл .ovpn надісланий.",
                parse_mode="HTML",
            )
        else:
            await update.message.reply_text(f"⚠ Файл
{client_name}.ovpn не знайдений.")
    except subprocess.CalledProcessError as e:
        await update.message.reply_text(f"❌ Помилка при додаванні
клієнта:\n{e}")


    await start(update, context)
    return ConversationHandler.END
```

Скрипт **add_user.sh** використовується для фактичного додавання нового клієнта до VPN. Він запускає внутрішній інсталяційний скрипт OpenVPN, передаючи ім'я клієнта як параметр. Після успішного виконання цього скрипту генерується файл конфігурації **.ovpn** для клієнта.

add_user.sh:

```
export MENU_OPTION="1"           # Вибір опції "додати клієнта" у головному меню інсталяційного скрипту

export CLIENT="$1"                # Ім'я клієнта передається як перший аргумент командного рядка

export PASS="1"                   # Встановлення пароля для нового клієнта

# Запуск основного скрипту OpenVPN

../openvpn-install.sh             # Виконання скрипту, який додає нового клієнта
```

Цей скрипт автоматизує процес додавання нового клієнта до VPN. Він використовує змінні середовища, щоб передати параметри до основного скрипту OpenVPN (`openvpn-install.sh`). Після запуску цей скрипт додає нового клієнта з ім'ям, переданим як аргумент.

3. Видалення клієнта VPN

Функція `remove_client_input` відкликає сертифікат клієнта, видаляє його конфігураційний файл та оновлює список відкликаних сертифікатів. Це забезпечує безпечне видалення клієнта з VPN.

```
async def remove_client_input(update: Update, context: ContextTypes.DEFAULT_TYPE):
    client_name = update.message.text


    try:
        cmd_revoke = f"cd {EASYRSA_PATH} && ./easyrsa --batch revoke {client_name} && ./easyrsa gen-crl"
        subprocess.run(cmd_revoke, shell=True, check=True)


        cmd_copy_crl = f"cp {EASYRSA_PATH}/pki/crl.pem /etc/openvpn/crl.pem"
        subprocess.run(cmd_copy_crl, shell=True, check=True)


        await update.message.reply_text(
```

```
            f"✅ Клієнт <b>{escape(client_name)}</b> успішно
видалений.",
            parse_mode="HTML",
        )

    except subprocess.CalledProcessError as e:
        await update.message.reply_text(f"❌ Помилка при видаленні
клієнта:\n{e}")

    await start(update, context)
    return ConversationHandler.END
```

4. Запуск сканування Nmap

Функція `start_nmap_scan` забезпечує запуск Nmap-сканування на Raspberry Pi через SSH. Результати сканування зберігаються у файлі на Raspberry Pi, після чого цей файл завантажується на сервер і надсилається користувачу через Telegram.

```
async def start_nmap_scan(update: Update, context:
ContextTypes.DEFAULT_TYPE, target: str):
    nmap_scan_type = context.user_data.get("nmap_scan_type", "full")
    scan_options = "-T4 -F" if nmap_scan_type == "quick" else "-A -
T4"
    command = f"nmap {scan_options} {target} -oN {remote_file_path}"

    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)
        stdin, stdout, stderr = ssh.exec_command(command)

        # Завантаження результатів після завершення сканування
        with SCPClient(ssh.get_transport()) as scp:
            scp.get(remote_file_path, local_file)

        with open(local_file, 'rb') as f:
```

```
            await update.message.reply_document(f,
filename=local_file)


    except Exception as e:

        await update.message.reply_text(f"❌ Помилка при запуску Nmap
сканування:\n{e}")


    await start(update, context)

    return ConversationHandler.END
```

5. Запуск сканування OpenVAS

Функція `start_openvas_scan` здійснює сканування OpenVAS, зберігаючи звіти у вигляді файлів XML і TXT. Функція здійснює підключення до сервера OpenVAS, створює завдання для сканування, моніторить виконання завдання і зберігає результати.

```
async def start_openvas_scan(update: Update, context:
ContextTypes.DEFAULT_TYPE, target: str):

    connection = TLSConnection(hostname=GMP_HOST, port=GMP_PORT)

    transform = EtreeTransform()

    with Gmp(connection, transform=transform) as gmp:

        gmp.authenticate(GMP_USERNAME, GMP_PASSWORD)


        task_id = gmp.create_task(...)  # створення задачі

        gmp.start_task(task_id)  # запуск задачі


        # Моніторинг і обробка результатів сканування

        ...
```

6. Перегляд і завантаження файлів сканування

Функції для перегляду та завантаження файлів сканування з Raspberry Pi забезпечують доступ до результатів, отриманих після проведення сканування Nmap або OpenVAS. Бот відображає список доступних файлів, дозволяючи завантажити вибраний файл безпосередньо через Telegram.

- Перегляд файлів сканування (`list_scans`) – відображає список файлів сканування, доступних для завантаження з Raspberry Pi.

- Завантаження результатів сканування (`download_scan`) – дозволяє завантажувати вибраний файл сканування.

7. Повний код Telegram-бота

```
import asyncio
import logging
import time
import os
import re
import subprocess
from datetime import datetime
import paramiko
from scp import SCPClient
from telegram import Update, InlineKeyboardButton,
InlineKeyboardMarkup
from telegram.ext import (
    ApplicationBuilder,
    CommandHandler,
    CallbackQueryHandler,
    ContextTypes,
    ConversationHandler,
    MessageHandler,
    filters,
)
from telegram.helpers import escape
import base64
from gvm.connections import TLSConnection
from gvm.protocols.gmp import Gmp
from gvm.transforms import EtreeTransform
from lxml import etree

# Logging settings
logging.basicConfig(
```

```
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
level=logging.INFO
)

# States for ConversationHandler
(
    ADDING_CLIENT,
    REMOVING_CLIENT,
    EXECUTING_COMMAND,
    DOWNLOADING_FILE,
    STARTING_OPENVAS_SCAN,
    CHOOSING_SCAN_TYPE,
    CHOOSING_NMAP_SCAN_TYPE,
    STARTING_NMAP_SCAN,
) = range(8)

# Telegram Token and ID of the authorized user
TOKEN = "TOKEN"
AUTHORIZED_USER_ID = ID

# Paths and settings
RASPBERRY_PI_VPN_IP = "IP"
SSH_USERNAME = "user"
SSH_PASSWORD = "pass"
GMP_HOST = RASPBERRY_PI_VPN_IP
GMP_PORT = 9390
GMP_USERNAME = 'admin'
GMP_PASSWORD = 'pass'
ADD_USER_SCRIPT = "/root/vpn_bot/add_user.sh"
OVPN_OUTPUT_DIR = "/root"
EASYRSA_PATH = "/etc/openvpn/easy-rsa"
```

```
# Directories for scan results on the server
SCAN_RESULTS_DIR = "/home/kali/scan_results/"
OPENVAS_RESULTS_DIR = "/home/kali/scan_results/openvas/"
NMAP_RESULTS_DIR = os.path.join(SCAN_RESULTS_DIR, "nmap/")
# Local path for temporary storage of results
LOCAL_RESULTS_DIR = "/tmp/openvas_results/"

# Function to display the main menu
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    logging.info("start command called")
    if update.effective_user.id != AUTHORIZED_USER_ID:
        await update.effective_message.reply_text("✖ У вас немає
доступу.")
        return ConversationHandler.END

    keyboard = [
        [InlineKeyboardButton("✚ Додати клієнта",
callback_data="add_client")],
        [InlineKeyboardButton("━ Видалити клієнта",
callback_data="remove_client")],
        [InlineKeyboardButton(" Активні клієнти",
callback_data="list_clients")],
        [InlineKeyboardButton(" Переглянути файли сканування",
callback_data="list_scans")],
        [InlineKeyboardButton(" Виконати команду",
callback_data="execute_command")],
        [InlineKeyboardButton(" Завантажити файл за шляхом",
callback_data="download_by_path")],
        [InlineKeyboardButton(" Запустити OpenVAS сканування",
callback_data="start_openvas_scan")],
        [InlineKeyboardButton(" Запустити Nmap сканування",
callback_data="start_nmap_scan")],
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
```

```
    await update.effective_message.reply_text(
        "□ Вітаю! Виберіть дію з меню нижче:",
reply_markup=reply_markup
    )
    return ConversationHandler.END


# Download file by path
async def download_by_path_input(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    file_path = update.message.text.strip()
    if file_path.lower() == "назад":
        return await go_back(update, context)


    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)


        with SCPClient(ssh.get_transport()) as scp:
            local_file = os.path.basename(file_path)
            scp.get(file_path, local_file)


        with open(local_file, "rb") as f:
            await update.message.reply_document(f,
filename=local_file)
        os.remove(local_file)
        ssh.close()


        await update.message.reply_text(f"✅ Файл {escape(file_path)}
успішно завантажений.", parse_mode="HTML")
    except Exception as e:
        logging.error(f"Error downloading file: {e}")
```

```
        await update.message.reply_text(f"❌ Помилка при завантаженні
файлу:\n{e}")


    await start(update, context)

    return ConversationHandler.END


# Function to go back to the main menu

async def go_back(update: Update, context:
ContextTypes.DEFAULT_TYPE):

    logging.info("go_back called")

    query = update.callback_query

    if query:

        await query.answer()

        await start(update, context)

    else:

        await start(update, context)

    return ConversationHandler.END


# Function to display the "Back" button

def back_button():

    keyboard = [[InlineKeyboardButton("□ Назад",
callback_data="go_back")]]

    return InlineKeyboardMarkup(keyboard)


# Handle commands from the menu

async def button_handler(update: Update, context:
ContextTypes.DEFAULT_TYPE):

    query = update.callback_query

    await query.answer()


    if update.effective_user.id != AUTHORIZED_USER_ID:

        await query.message.reply_text("❌ У вас немає доступу.")

        return
```

```
    data = query.data
    logging.info(f"Button pressed: {data}")

    if data == "add_client":
        await query.message.reply_text(
            "➖ Введіть ім'я нового клієнта або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return ADDING_CLIENT
    elif data == "remove_client":
        await query.message.reply_text(
            "➖ Введіть ім'я клієнта для видалення або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return REMOVING_CLIENT
    elif data == "list_clients":
        await list_clients_command(update, context)
    elif data == "list_scans":
        await list_scans(update, context)
    elif data == "download_by_path":
        await query.message.reply_text(
            "➖ Введіть шлях до файлу для завантаження або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return DOWNLOADING_FILE
    elif data == "execute_command":
        await query.message.reply_text(
            "➖ Введіть команду для виконання на сервері або натисніть 'Назад', щоб повернутися:",
```

```
            reply_markup=back_button(),
        )
        return EXECUTING_COMMAND
    elif data == "start_openvas_scan":
        await choose_scan_type(update, context)
        return CHOOSING_SCAN_TYPE
    elif data == "start_nmap_scan":
        await choose_nmap_scan_type(update, context)
        return CHOOSING_NMAP_SCAN_TYPE
    elif data == "quick_scan":
        context.user_data["scan_type"] = "quick_scan"
        await query.message.reply_text(
            "➖ Введіть ціль для OpenVAS сканування (IP або діапазон)
або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return STARTING_OPENVAS_SCAN
    elif data == "full_scan":
        context.user_data["scan_type"] = "full_scan"
        await query.message.reply_text(
            "➖ Введіть ціль для OpenVAS сканування (IP або діапазон)
або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return STARTING_OPENVAS_SCAN
    elif data == "nmap_quick_scan":
        context.user_data["nmap_scan_type"] = "quick"
        await query.message.reply_text(
            "➖ Введіть ціль для Nmap сканування (IP або діапазон)
або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return STARTING_NMAP_SCAN
```

```
    elif data == "nmap_full_scan":
        context.user_data["nmap_scan_type"] = "full"
        await query.message.reply_text(
            "➖ Введіть ціль для Nmap сканування (IP або діапазон) або натисніть 'Назад', щоб повернутися:",
            reply_markup=back_button(),
        )
        return STARTING_NMAP_SCAN
    elif data == "go_back":
        await go_back(update, context)
    elif data in context.user_data.get('file_mappings', {}):
        await download_scan(update, context, data)
    else:
        await query.message.reply_text("❌ Невідома команда.")
    return ConversationHandler.END


# Function to choose OpenVAS scan type
async def choose_scan_type(update: Update, context: ContextTypes.DEFAULT_TYPE):
    keyboard = [
        [InlineKeyboardButton("Швидке сканування", callback_data="quick_scan")],
        [InlineKeyboardButton("Повне сканування", callback_data="full_scan")],
        [InlineKeyboardButton("□ Назад", callback_data="go_back")],
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.callback_query.message.reply_text("Виберіть тип сканування OpenVAS:", reply_markup=reply_markup)


# Function to choose Nmap scan type
async def choose_nmap_scan_type(update: Update, context: ContextTypes.DEFAULT_TYPE):
    keyboard = [
```

```
        [InlineKeyboardButton("Швидке сканування",
callback_data="nmap_quick_scan")],
        [InlineKeyboardButton("Повне сканування",
callback_data="nmap_full_scan")],
        [InlineKeyboardButton("□ Назад", callback_data="go_back")],
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.callback_query.message.reply_text("Виберіть тип
сканування Nmap:", reply_markup=reply_markup)


# Add client
async def add_client_input(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    client_name = update.message.text
    ovpn_file = f"{OVPN_OUTPUT_DIR}/{client_name}.ovpn"


    try:
        cmd_create = f"sudo bash {ADD_USER_SCRIPT} {client_name}"
        subprocess.run(cmd_create, shell=True, check=True)


        if os.path.exists(ovpn_file):
            with open(ovpn_file, "rb") as f:
                await update.message.reply_document(f,
filename=f"{client_name}.ovpn")
            os.remove(ovpn_file)


            await update.message.reply_text(
                f"✅ Клієнт <b>{escape(client_name)}</b> успішно
доданий, файл .ovpn надісланий.",
                parse_mode="HTML",
            )
        else:
            await update.message.reply_text(f"⚠ Файл
{client_name}.ovpn не знайдений.")
```

```
        # Cleanup revoked certificates
        cleanup_revoked_certs()

    except subprocess.CalledProcessError as e:
        logging.error(f"Error adding client: {e}")
        await update.message.reply_text(f"❌ Помилка при додаванні клієнта:\n{e}")
    except Exception as e:
        logging.error(f"Unexpected error: {e}")
        await update.message.reply_text(f"❌ Непередбачена помилка:\n{e}")

    await start(update, context)
    return ConversationHandler.END

# Remove client
async def remove_client_input(update: Update, context: ContextTypes.DEFAULT_TYPE):
    client_name = update.message.text

    try:
        cmd_revoke = f"cd {EASYRSA_PATH} && ./easyrsa --batch revoke {client_name} && ./easyrsa gen-crl"
        subprocess.run(cmd_revoke, shell=True, check=True)

        cmd_copy_crl = f"cp {EASYRSA_PATH}/pki/crl.pem /etc/openvpn/crl.pem"
        subprocess.run(cmd_copy_crl, shell=True, check=True)

        client_config = f"/etc/openvpn/client-configs/files/{client_name}.ovpn"
        if os.path.exists(client_config):
            os.remove(client_config)
```

```
        await update.message.reply_text(
            f"✅ Клієнт <b>{escape(client_name)}</b> успішно видалений.",
            parse_mode="HTML",
        )

        # Cleanup revoked certificates
        cleanup_revoked_certs()

    except subprocess.CalledProcessError as e:
        logging.error(f"Error removing client: {e}")
        await update.message.reply_text(f"❌ Помилка при видаленні клієнта:\n{e}")
    except Exception as e:
        logging.error(f"Unexpected error: {e}")
        await update.message.reply_text(f"❌ Непередбачена помилка:\n{e}")

    await start(update, context)
    return ConversationHandler.END

# Function to clean up revoked certificates
def cleanup_revoked_certs():
    index_file = f"{EASYRSA_PATH}/pki/index.txt"
    with open(index_file, 'r') as f:
        lines = f.readlines()

    active_lines = []
    for line in lines:
        if line.startswith('V'):
            active_lines.append(line)
        else:
```

```
            # Remove revoked certificate files
            parts = line.strip().split('\t')
            if len(parts) >= 6:
                dn = parts[5]
                cn = dn.split('CN=')[-1]
                cert_file = f"{EASYRSA_PATH}/pki/issued/{cn}.crt"
                key_file = f"{EASYRSA_PATH}/pki/private/{cn}.key"
                req_file = f"{EASYRSA_PATH}/pki/reqs/{cn}.req"
                # Remove files if they exist
                for file_path in [cert_file, key_file, req_file]:
                    if os.path.exists(file_path):
                        os.remove(file_path)

    # Rewrite index.txt with active certificates
    with open(index_file, 'w') as f:
        f.writelines(active_lines)

    # Update CRL
    subprocess.run(f"cd {EASYRSA_PATH} && ./easyrsa gen-crl",
shell=True, check=True)
    subprocess.run(f"cp {EASYRSA_PATH}/pki/crl.pem
/etc/openvpn/crl.pem", shell=True, check=True)

# Execute commands on the server
async def execute_command_input(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    command = update.message.text

    try:
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)
```

```
        stdin, stdout, stderr = ssh.exec_command(command)
        result = stdout.read().decode()
        error = stderr.read().decode()
        ssh.close()

        if result:
            await update.message.reply_text(f"✅ <b>Результат
виконання:</b>\n{escape(result)}", parse_mode="HTML")
        if error:
            await update.message.reply_text(f"❌
<b>Помилка:</b>\n{escape(error)}", parse_mode="HTML")
    except Exception as e:
        logging.error(f"Error executing command: {e}")
        await update.message.reply_text(f"❌ Помилка при виконанні
команди:\n{e}")

    await start(update, context)
    return ConversationHandler.END

# Function to extract and decode Base64 content
def extract_and_decode_base64(file_path, output_path):
    """Extracts and decodes Base64-encoded data from an XML
report."""
    import re
    import base64

    # Read XML as text
    with open(file_path, 'r', encoding='utf-8') as file:
        xml_content = file.read()

    # Search for Base64 data between </report_format> and </report>
    base64_pattern =
re.search(r'</report_format>([\s\S]*?)</report>', xml_content)
```

```
    if base64_pattern:
        base64_data = base64_pattern.group(1).strip()

        try:
            # Decode Base64
            decoded_text =
base64.b64decode(base64_data).decode('utf-8')

            # Write decoded text to output file
            with open(output_path, 'w', encoding='utf-8') as
output_file:
                output_file.write(decoded_text)

            print("Дані успішно декодовані та записані у файл.")
            return True  # Successfully decoded and written
        except (base64.binascii.Error, UnicodeDecodeError) as e:
            print(f"Помилка декодування Base64: {e}. Перевірте
формат даних.")
            return False
    else:
        print("Поле з Base64 не знайдено між </report_format> і
</report>.")
        return False

# Function to start OpenVAS scan and send the report
# Function to start OpenVAS scan and send the report
async def start_openvas_scan(update: Update, context:
ContextTypes.DEFAULT_TYPE, target: str):
    scan_type = context.user_data.get("scan_type", "full_scan")
    config_name = "Full and fast" if scan_type == "full_scan" else
"Host Discovery"

    # Create unique filenames based on time and target
```

```
    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
    safe_target = re.sub(r'[^\w\-_. ]', '_', target)
    base_filename = f"openvas_scan_{safe_target}_{timestamp}"
    xml_filename = f"{base_filename}.xml"
    txt_filename = f"{base_filename}.txt"

    # Full paths to files in local directory
    xml_file_path = os.path.join(LOCAL_RESULTS_DIR, xml_filename)
    txt_file_path = os.path.join(LOCAL_RESULTS_DIR, txt_filename)

    # Create local directory for results
    os.makedirs(LOCAL_RESULTS_DIR, exist_ok=True)

    try:
        # Connect to OpenVAS
        connection = TLSConnection(hostname=GMP_HOST, port=GMP_PORT)
        transform = EtreeTransform()
        with Gmp(connection, transform=transform) as gmp:
            gmp.authenticate(GMP_USERNAME, GMP_PASSWORD)

            # Get scan configuration
            configs = gmp.get_scan_configs()
            config_id = None
            for config in configs.xpath('.//config'):
                if config.find('name').text == config_name:
                    config_id = config.get('id')
                    break
            if not config_id:
                await update.message.reply_text(f"❌ Не вдалося
знайти конфігурацію '{config_name}'.")
                return
```

```
            # Get scanner ID for OpenVAS
            scanners = gmp.get_scanners()
            scanner_id = None
            for scanner in scanners.xpath('.//scanner'):
                if "OpenVAS" in scanner.find('name').text:
                    scanner_id = scanner.get('id')
                    break
            if not scanner_id:
                await update.message.reply_text("❌ Не вдалося знайти
сканер OpenVAS.")
                return

            # Get port list ID
            port_lists = gmp.get_port_lists()
            port_list_name = 'All TCP and Nmap top 100 UDP'  #
Adjust this name as needed
            port_list_id = None
            for port_list in port_lists.xpath('.//port_list'):
                if port_list.find('name').text == port_list_name:
                    port_list_id = port_list.get('id')
                    break
            if not port_list_id:
                await update.message.reply_text(f"❌ Не вдалося
знайти порт-лист '{port_list_name}'.")
                return

            # Check if target already exists
            existing_target_id = None
            targets = gmp.get_targets()
            for t in targets.xpath('.//target'):
                if target in t.find('hosts').text:
                    existing_target_id = t.get('id')
                    break
```

```
            # Use existing target or create a new one
            if existing_target_id:
                target_id = existing_target_id
                await update.message.reply_text(f"✅ Використовуємо
наявну ціль з ID {target_id} для {target}.")
            else:
                target_resp = gmp.create_target(
                    name=f'Target {target}',
                    hosts=[target],
                    port_list_id=port_list_id
                )
                target_id = target_resp.get('id')
                if not target_id:
                    status_text =
target_resp.xpath('//@status_text')[0]
                    await update.message.reply_text(f"❌ Не вдалося
створити ціль для {target}. Статус: {status_text}")
                    return

            # Create and start the task
            task_resp = gmp.create_task(
                name=f'Task {target}',
                config_id=config_id,
                target_id=target_id,
                scanner_id=scanner_id
            )
            task_id = task_resp.get('id')
            if not task_id:
                await update.message.reply_text(f"❌ Не вдалося
створити задачу для цілі {target}.")
                return
```

```
            gmp.start_task(task_id)
            await update.message.reply_text(f"✅ Сканування цілі
{target} запущено ({config_name}).")

            # Monitor scan progress
            status = ''
            previous_progress = -1
            while status != 'Done':
                await asyncio.sleep(30)
                task = gmp.get_task(task_id=task_id)
                status = task.find('.//status').text
                progress_element = task.find('.//progress')
                if progress_element is not None:
                    progress = int(progress_element.text)
                else:
                    progress = None

                if progress != previous_progress and progress is not
None:
                    await update.message.reply_text(f" Прогрес
сканування: {progress}%")
                    previous_progress = progress

            await update.message.reply_text("✅ Сканування завершено.
Формуємо звіт...")

            # Get the report in XML format
            report_id = task.find('.//last_report/report').get('id')
            txt_format_id = 'a3810a62-1f62-11e1-9219-406186ea4fc5'
# ID for TXT format
            xml_format_id = 'a994b278-1f62-11e1-96ac-406186ea4fc5'
# ID for XML format

            # Save the report to a local XML file
```

```
            report_resp_xml = gmp.get_report(report_id=report_id,
report_format_id=xml_format_id)
            with open(xml_file_path, 'wb') as f:
                f.write(etree.tostring(report_resp_xml,
pretty_print=True, encoding='utf-8', xml_declaration=True))


            # Get and decode the report in TXT format
            report_resp_txt = gmp.get_report(report_id=report_id,
report_format_id=txt_format_id)
            temp_xml_file = f"/tmp/{base_filename}_temp.xml"
            with open(temp_xml_file, 'wb') as f:
                f.write(etree.tostring(report_resp_txt,
pretty_print=True, encoding='utf-8', xml_declaration=True))


            # Decode Base64 content and save as a text file
            success = extract_and_decode_base64(temp_xml_file,
txt_file_path)
            os.remove(temp_xml_file)  # Remove the temporary file


            if success:
                # Send files to the user
                with open(txt_file_path, 'rb') as f:
                    await update.message.reply_document(f,
filename=os.path.basename(txt_file_path))
                with open(xml_file_path, 'rb') as f:
                    await update.message.reply_document(f,
filename=os.path.basename(xml_file_path))
            else:
                await update.message.reply_text("✗ Не вдалося
витягти та декодувати Base64-контент із XML-звіту.")


            # Upload files to Raspberry Pi
            ssh = paramiko.SSHClient()

ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
```

```
            ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)
            ssh.exec_command(f"mkdir -p {OPENVAS_RESULTS_DIR}")  #
Ensure directory exists
            with SCPClient(ssh.get_transport()) as scp:
                scp.put(txt_file_path,
os.path.join(OPENVAS_RESULTS_DIR, os.path.basename(txt_file_path)))
                scp.put(xml_file_path,
os.path.join(OPENVAS_RESULTS_DIR, os.path.basename(xml_file_path)))
            ssh.close()

            # Clean up local files after transfer
            os.remove(txt_file_path)
            os.remove(xml_file_path)

            await update.message.reply_text("✅ Звіти успішно
збережені на Raspberry Pi.")

            # Optionally delete the task
            gmp.delete_task(task_id)

    except Exception as e:
        logging.error(f"Error starting OpenVAS scan: {e}")
        await update.message.reply_text(f"❌ Помилка при запуску
OpenVAS сканування:\n{e}")

# Handle input for starting OpenVAS scan
async def start_openvas_scan_input(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    target = update.message.text.strip()
    if target.lower() == "назад":
        return await go_back(update, context)

    logging.info(f"Starting OpenVAS scan on target: {target}")
```

```
    await update.message.reply_text(f"🔍 Починаємо OpenVAS сканування цілі: {target}")

    await start_openvas_scan(update, context, target)

    await start(update, context)
    return ConversationHandler.END

# List active clients
async def list_clients_command(update: Update, context: ContextTypes.DEFAULT_TYPE):
    try:
        index_file = f"{EASYRSA_PATH}/pki/index.txt"
        with open(index_file, 'r') as f:
            lines = f.readlines()

        active_clients = []
        for line in lines:
            if line.startswith('V'):
                parts = line.strip().split('\t')
                if len(parts) >= 6:
                    dn = parts[5]
                    cn = dn.split('CN=')[-1]
                    active_clients.append(cn)

        if active_clients:
            clients_list = '\n'.join(active_clients)
            await update.callback_query.message.reply_text(
                f"📋 <b>Список активних клієнтів:</b>\n{escape(clients_list)}", parse_mode="HTML"
            )
        else:
```

```
            await update.callback_query.message.reply_text("⚠ Немає
активних клієнтів.")
    except Exception as e:
        logging.error(f"Error listing clients: {e}")
        await update.callback_query.message.reply_text(f"✖ Помилка
при отриманні списку клієнтів:\n{e}")


# List scan files on Raspberry Pi
async def list_scans(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    try:
        # Initialize file mappings
        context.user_data['file_mappings'] = {}
        file_index = 0


        # Set up SSH connection to Raspberry Pi
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)


        # Specify paths for OpenVAS and Nmap on Raspberry Pi
        scan_dirs = {"OpenVAS": OPENVAS_RESULTS_DIR, "Nmap":
NMAP_RESULTS_DIR}
        files = []


        # Get the list of files for each scan type
        for scan_type, dir_path in scan_dirs.items():
            stdin, stdout, stderr = ssh.exec_command(f"ls
{dir_path}")
            dir_files = stdout.read().decode().splitlines()


            # Add files to the list if they exist
            for f in dir_files:
```

```
                file_id = str(file_index)
                files.append((file_id, scan_type, f))
                context.user_data['file_mappings'][file_id] =
(scan_type, f)
                file_index += 1


        ssh.close()


        # Create keyboard with the files found
        if files:
            keyboard = []
            for file_id, scan_type, file_name in sorted(files):
                display_name = f"{scan_type}: {file_name}"
                keyboard.append([InlineKeyboardButton(display_name,
callback_data=file_id)])
            keyboard.append([InlineKeyboardButton("□ Назад",
callback_data="go_back")])
            reply_markup = InlineKeyboardMarkup(keyboard)
            await update.callback_query.message.reply_text("□
Виберіть файл для завантаження:", reply_markup=reply_markup)
        else:
            await update.callback_query.message.reply_text("⚠ Немає
доступних файлів сканування.")
    except Exception as e:
        logging.error(f"Error listing scans: {e}")
        await update.callback_query.message.reply_text(f"✕ Помилка
при отриманні списку сканів:\n{e}")


# Download scan file from Raspberry Pi
async def download_scan(update: Update, context:
ContextTypes.DEFAULT_TYPE, file_id: str):
    try:
        if file_id not in context.user_data.get('file_mappings',
{}):
```

```
            await update.callback_query.message.reply_text("❌
Невідомий файл.")
            return

        scan_type, file_name =
context.user_data['file_mappings'][file_id]
        dir_path = OPENVAS_RESULTS_DIR if scan_type == "OpenVAS"
else NMAP_RESULTS_DIR
        remote_file_path = os.path.join(dir_path, file_name)

        # Connect to Raspberry Pi via SSH
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)

        # Download the file from Raspberry Pi
        with SCPClient(ssh.get_transport()) as scp:
            local_file = os.path.basename(remote_file_path)
            scp.get(remote_file_path, local_file)

        # Send the file to the user
        with open(local_file, "rb") as f:
            await update.callback_query.message.reply_document(f,
filename=file_name)

        # Close SSH connection and remove local file
        ssh.close()
        os.remove(local_file)

    except Exception as e:
        logging.error(f"Error downloading scan: {e}")
        await update.callback_query.message.reply_text(f"❌ Помилка
при завантаженні файлу:\n{e}")
```

```
    await start(update, context)


# Function to start Nmap scan via SSH on Raspberry Pi
async def start_nmap_scan(update: Update, context:
ContextTypes.DEFAULT_TYPE, target: str):
    nmap_scan_type = context.user_data.get("nmap_scan_type", "full")
    safe_target = re.sub(r'[^\w\-_. ]', '_', target)

    # Create unique filename based on time and target
    timestamp = datetime.now().strftime("%Y%m%d%H%M%S")
    base_filename = f"nmap_scan_{safe_target}_{timestamp}.txt"
    remote_file_path = os.path.join(NMAP_RESULTS_DIR, base_filename)

    # Select scan options
    if nmap_scan_type == "quick":
        scan_options = "-T4 -F"
    else:  # full scan
        scan_options = "-A -T4"

    try:

        # Connect to Raspberry Pi via SSH
        ssh = paramiko.SSHClient()
        ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        ssh.connect(RASPBERRY_PI_VPN_IP, username=SSH_USERNAME,
password=SSH_PASSWORD)

        # Ensure the results directory exists
        stdin, stdout, stderr = ssh.exec_command(f"mkdir -p
{NMAP_RESULTS_DIR}")
        stderr_output = stderr.read().decode()
        if stderr_output:
```

```
            logging.error(f"Error creating directory on Raspberry
Pi: {stderr_output}")


        # Execute Nmap command on Raspberry Pi
        command = f"nmap {scan_options} {target} -oN
{remote_file_path}"
        stdin, stdout, stderr = ssh.exec_command(command)


        # Wait for the scan to complete
        exit_status = stdout.channel.recv_exit_status()
        if exit_status != 0:
            error_output = stderr.read().decode()
            await update.message.reply_text(f"❌ Помилка при
виконанні Nmap сканування:\n{error_output}")
            ssh.close()
            return


        # Download the scan result
        sftp = ssh.open_sftp()
        local_file = os.path.basename(remote_file_path)
        sftp.get(remote_file_path, local_file)
        sftp.close()
        ssh.close()


        # Send the scan result to the user
        with open(local_file, 'rb') as f:
            await update.message.reply_document(f,
filename=local_file)


        # Remove the local file
        os.remove(local_file)


        await update.message.reply_text(f"✅ Nmap сканування цілі
{target} завершено.")
```

```
    except Exception as e:
        logging.error(f"Error starting Nmap scan: {e}")
        await update.message.reply_text(f"❌ Помилка при запуску Nmap сканування:\n{e}")

# Handle input for starting Nmap scan
async def start_nmap_scan_input(update: Update, context: ContextTypes.DEFAULT_TYPE):
    target = update.message.text.strip()
    if target.lower() == "назад":
        return await go_back(update, context)

    logging.info(f"Starting Nmap scan on target: {target}")
    await update.message.reply_text(f"🔍 Починаємо Nmap сканування цілі: {target}")

    await start_nmap_scan(update, context, target)

    await start(update, context)
    return ConversationHandler.END

# Main function to run the bot
def main():
    application = ApplicationBuilder().token(TOKEN).build()

    conv_handler = ConversationHandler(
        entry_points=[
            CommandHandler("start", start),
            CallbackQueryHandler(button_handler),
        ],
        states={
```

```
            ADDING_CLIENT: [MessageHandler(filters.TEXT &
~filters.COMMAND, add_client_input)],

            REMOVING_CLIENT: [MessageHandler(filters.TEXT &
~filters.COMMAND, remove_client_input)],

            EXECUTING_COMMAND: [MessageHandler(filters.TEXT &
~filters.COMMAND, execute_command_input)],

            DOWNLOADING_FILE: [MessageHandler(filters.TEXT &
~filters.COMMAND, download_by_path_input)],

            STARTING_OPENVAS_SCAN: [MessageHandler(filters.TEXT &
~filters.COMMAND, start_openvas_scan_input)],

            CHOOSING_SCAN_TYPE:
[CallbackQueryHandler(button_handler)],

            CHOOSING_NMAP_SCAN_TYPE:
[CallbackQueryHandler(button_handler)],

            STARTING_NMAP_SCAN: [MessageHandler(filters.TEXT &
~filters.COMMAND, start_nmap_scan_input)],

        },

        fallbacks=[CallbackQueryHandler(go_back)],

    )


    application.add_handler(conv_handler)

    application.run_polling()


if __name__ == "__main__":

    main()
```

2.6 Налаштування вразливої машини для тестування внутрішньої мережі

Для створення безпечного середовища тестування на вразливості в мережі було налаштовано вразливу машину **Metasploitable3**, яка працюватиме у віртуальному середовищі VirtualBox на окремому ноутбуці. Metasploitable3 імітує реальне середовище з типовими проблемами безпеки, що дозволяє продемонструвати здатність Raspberry Pi до взаємодії з різними хостами в локальній мережі та збору інформації про них. Це також показує, як Raspberry Pi може отримати доступ до будь-якого хоста, розташованого в тій самій мережі.

2.6.1 Завантаження та налаштування Metasploitable3

Metasploitable3 — це спеціально створений вразливий образ, що містить численні відомі вразливості. Налаштування цього образу у вигляді віртуальної машини забезпечує повноцінне середовище для тестування.

Кроки для завантаження та розгортання Metasploitable3:

1. Завантаження образу Metasploitable3:

Завантаження образу здійснюється, перейшовши на сайт SourceForge і в пошуку знайти Metasploitable3 [13], після чого завантажити образ.

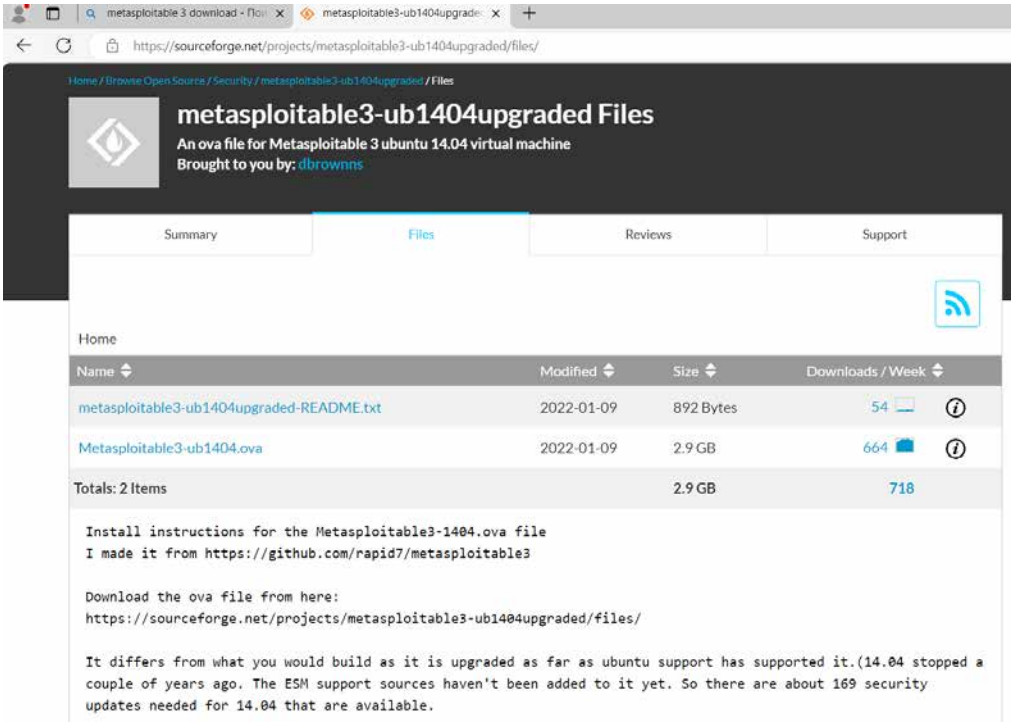


Рисунок 2.29 – Завантаження образу Metasploitable3

2. Розгортання віртуальної машини на VirtualBox:

У програмі **VirtualBox** на окремому ноутбуці здійснюється імпорт завантаженого образу Metasploitable3 як нової віртуальної машини з відповідними параметрами.

3. Налаштування мережевого адаптера в VirtualBox:

У налаштуваннях мережі для віртуальної машини обирається режим **Bridged Adapter**. Це дозволяє віртуальній машині отримати окрему IP-адресу в локальній мережі ноутбука, що забезпечує її доступність для Raspberry Pi, підключеного до тієї ж мережі.

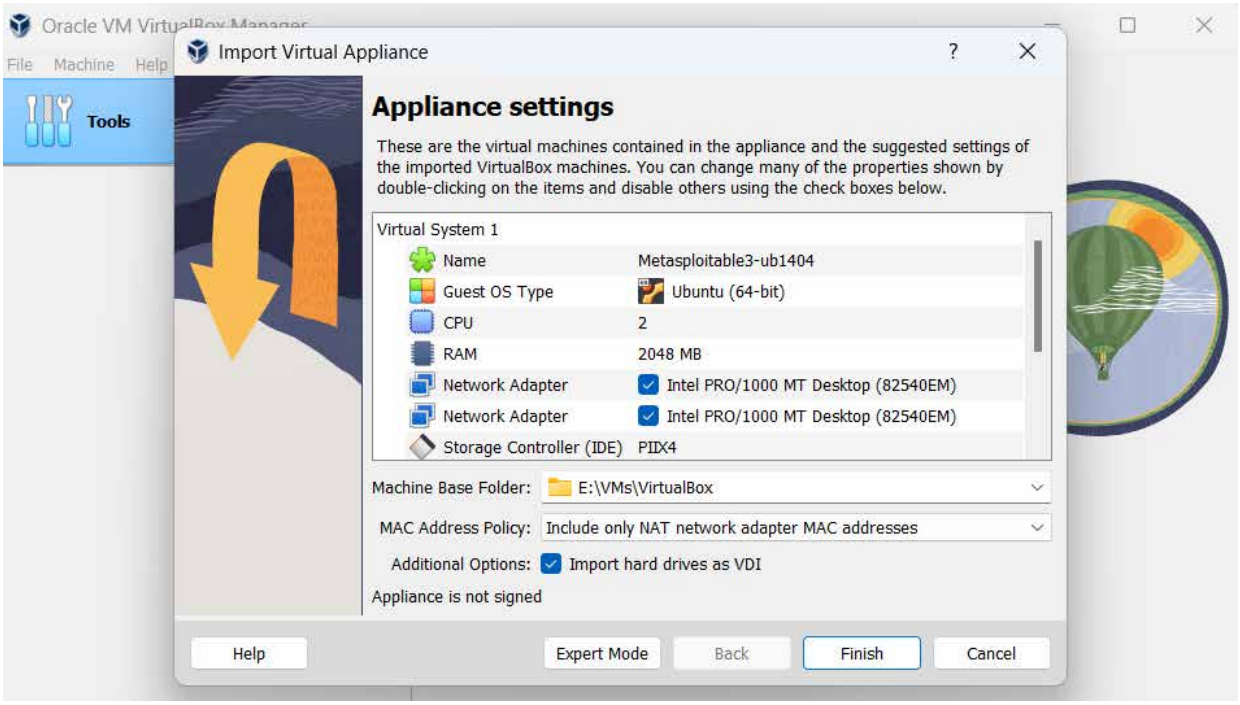


Рисунок 2.30 – Імпорт образу Metasploitable3

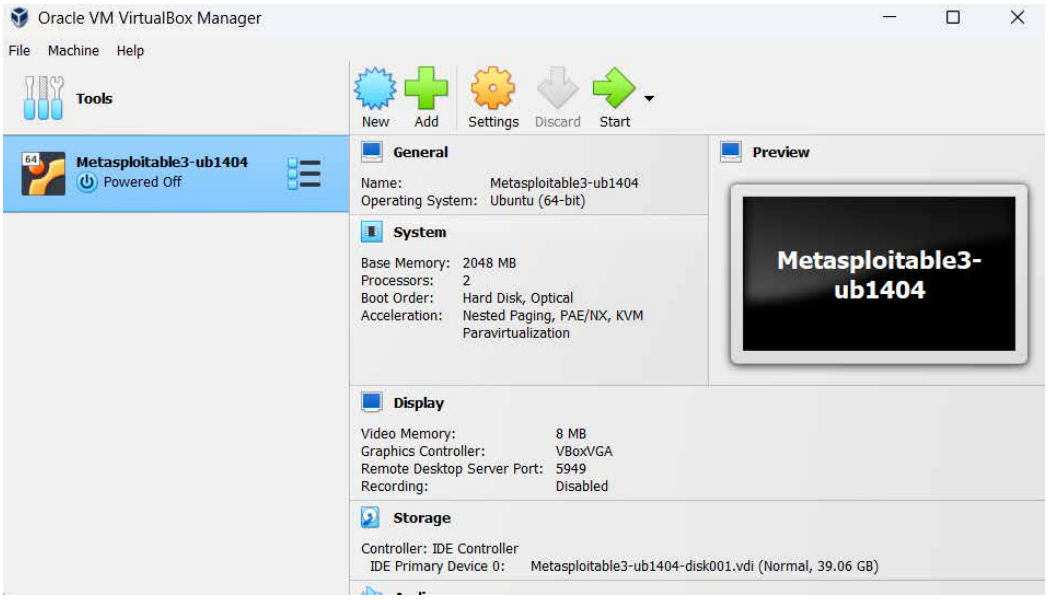


Рисунок 2.31 – Встановлений образ Metasploitable3

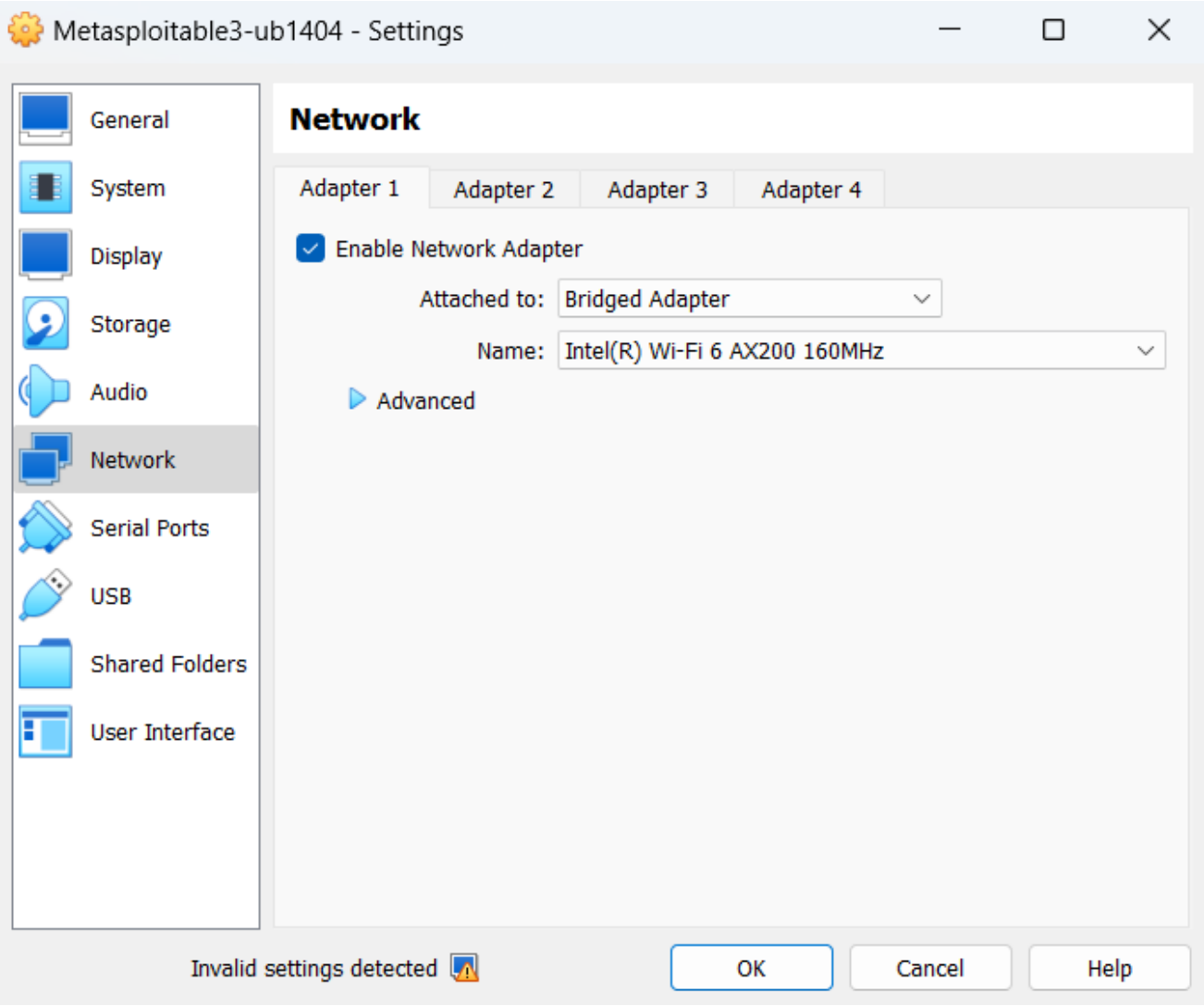


Рисунок 2.31 – Налаштування мережевого адаптера

3 ПОБУДОВА МОДЕЛІ ЗАХИЩЕНОЇ МЕРЕЖІ В НАВЧАЛЬНОМУ ЗАКЛАДІ

3.1 Запуск сканування через Telegram-боту

Telegram-бот, розроблений у попередньому розділі, надає інтерфейс для віддаленого запуску сканувань, управління процесом та отримання результатів. Використання бота дозволяє не лише автоматизувати процес сканування, а й забезпечує зручність у контролі, особливо під час тестування в умовах обмежених ресурсів, таких як на Raspberry Pi.

1. Запуск сканування Nmap через Telegram-бот:

У Telegram-боті для виконання сканування з Nmap використовується команда **/start_nmap_scan**.

Після вибору типу сканування (швидке або повне) бот запитує цільову IP-адресу або діапазон для сканування. Введення IP-адреси Metasploitable3 дозволяє боту автоматично розпочати сканування і записати результати в заздалегідь визначений файл.

По завершенню сканування бот надсилає файл з результатами сканування безпосередньо у чат, що дозволяє швидко переглянути основні дані.

2. Запуск сканування OpenVAS через Telegram-бот:

Для виконання сканування з OpenVAS використовується команда **/start_openvas_scan** у Telegram-боті.

Після вибору типу сканування (швидке або повне) бот запитує ціль для сканування. OpenVAS виконує глибокий аналіз вразливостей на вказаній цілі (IP-адреса Metasploitable3), а результати зберігаються у вигляді текстового та XML-звіту.

По завершенню сканування бот надсилає згенерований звіт у чат, що надає детальну інформацію про знайдені вразливості.

3.2 Порівняння результатів сканування

Після завершення сканування за допомогою Nmap та OpenVAS було проведено порівняння отриманих результатів, що дозволяє оцінити ефективність кожного інструменту.

1. Nmap:

- Результати сканування Nmap містять інформацію про відкриті порти, виявлені сервіси, версії програмного забезпечення та базові операційні системи.
- Nmap забезпечує швидке сканування з базовою інформацією про ціль, що підходить для початкового аналізу мережі.

2. OpenVAS:

- Результати OpenVAS надають глибокий аналіз вразливостей, включаючи конкретні ідентифікатори CVE (Common Vulnerabilities and Exposures) та детальний опис виявлених проблем безпеки.
- OpenVAS дозволяє оцінити критичність вразливостей та забезпечує рекомендації щодо усунення проблем, що робить його корисним для більш поглибленого аналізу.

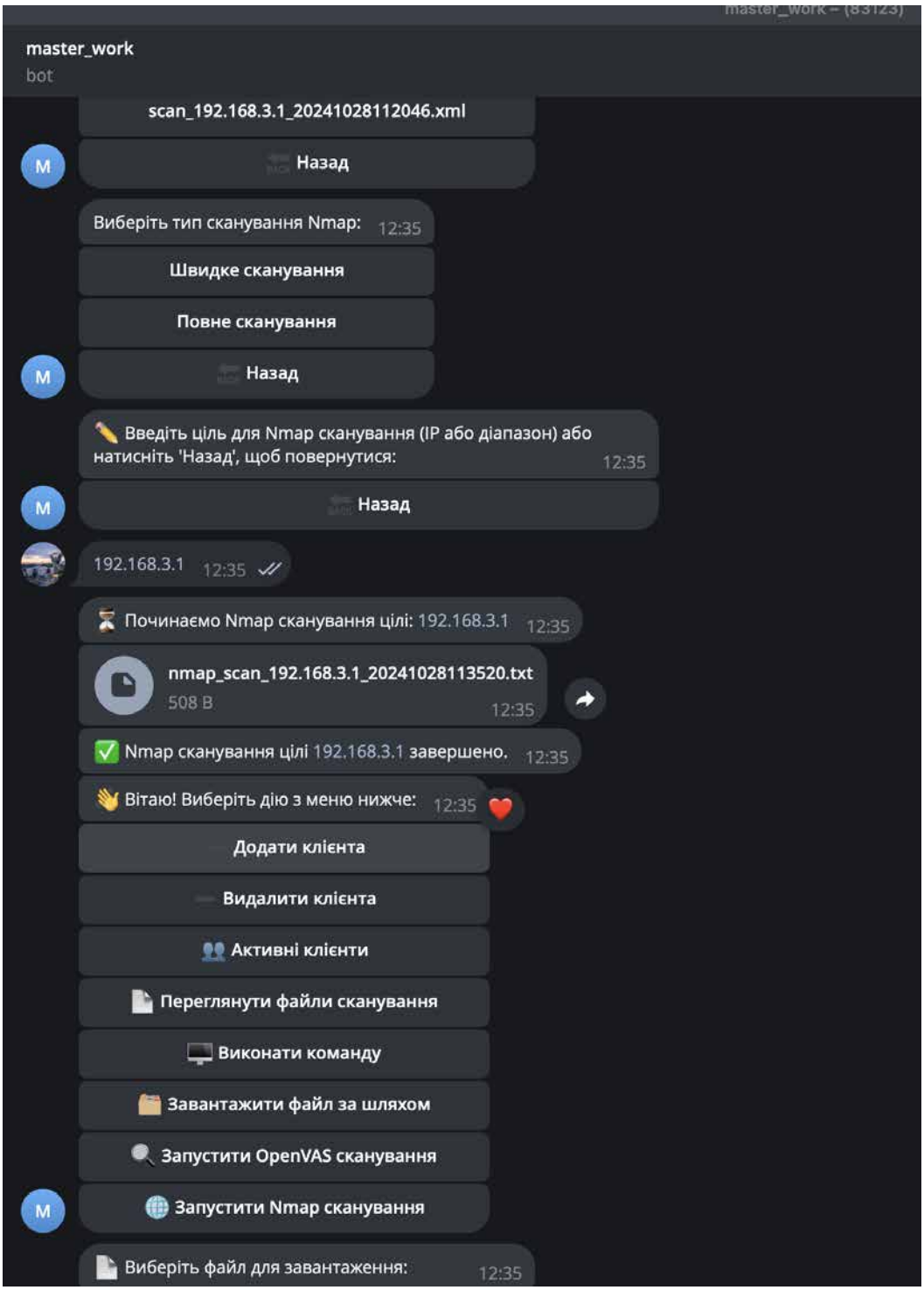


Рисунок 3.1 – Сканування за допомогою nmap

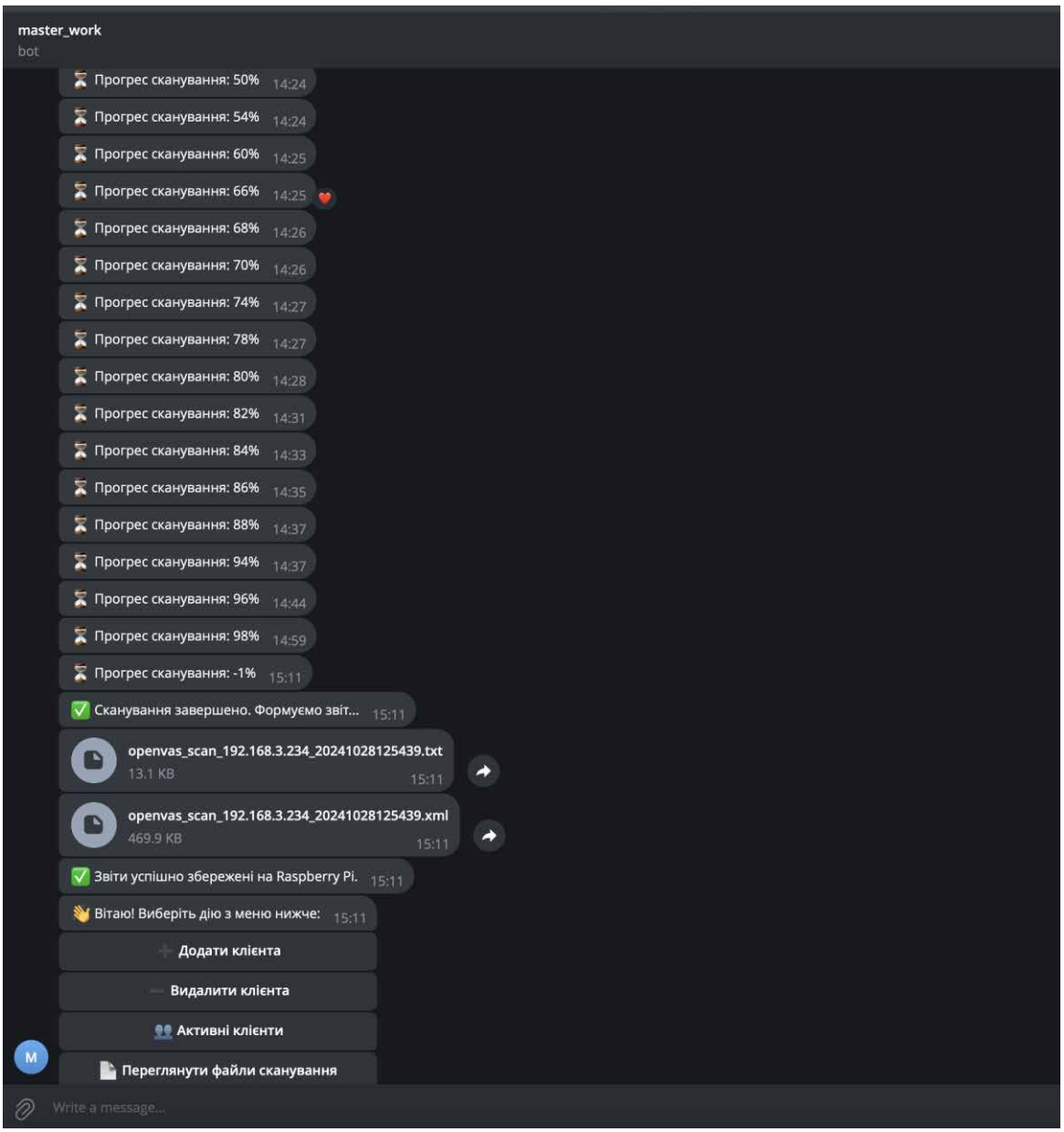


Рисунок 3.2 – Повне сканування за допомогою OpenVAS

3.3 Аналіз продуктивності та навантаження на систему

Оцінка продуктивності включає швидкість сканування, рівень навантаження на Raspberry Pi, а також точність результатів.

1. Швидкість сканування:

- Nmap: Завдяки своїм оптимізованим алгоритмам Nmap забезпечує швидке сканування, яке зазвичай триває від 2 до 5 хвилин, залежно від кількості відкритих портів на цільовій машині.
- OpenVAS: Сканування OpenVAS займає значно більше часу, оскільки виконується глибокий аналіз кожного сервісу. Залежно від налаштувань і кількості відкритих портів, час сканування може становити 20–30 хвилин і більше.

2. Навантаження на систему:

- Nmap: Під час сканування навантаження на CPU і RAM Raspberry Pi залишається на низькому рівні, що робить цей інструмент придатним для швидкого аналізу без значного впливу на продуктивність.
- OpenVAS: Через використання складних алгоритмів і бази даних з відомими вразливостями навантаження на Raspberry Pi під час сканування OpenVAS є значно вищим. Цей сканер потребує більше оперативної пам'яті та процесорних ресурсів, який може вплинути на загальну стабільність системи при тривалих скануваннях.

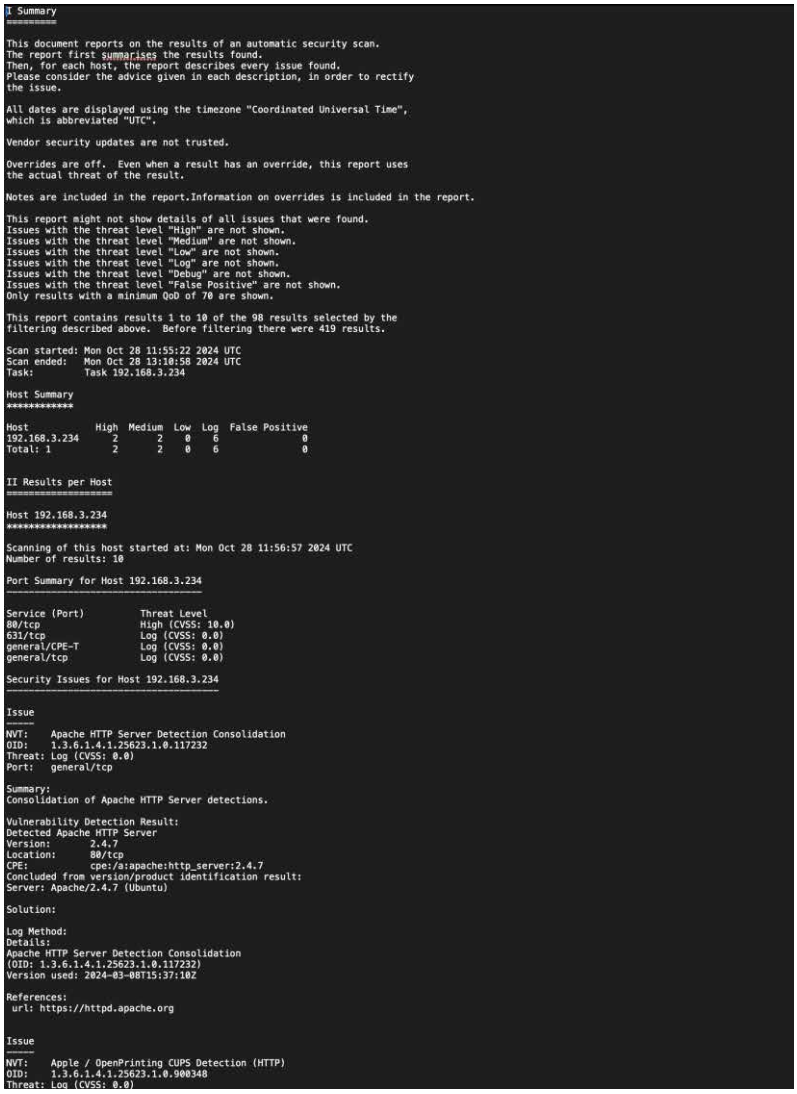
3. Точність та обсяг отриманої інформації:

- Nmap: Забезпечує основні дані про сервіси та порти, але має обмеження у виявленні специфічних вразливостей.
- OpenVAS: Надає детальну інформацію про вразливості та ризики, включаючи критичність і рекомендації з усунення, що робить його більш точним для комплексного аналізу безпеки.

3.4 Загальний аналіз безпеки та оцінка результатів

1. Точність виявлених вразливостей:

OpenVAS показує кращу точність у виявленні конкретних вразливостей завдяки доступу до бази CVE та здатності аналізувати внутрішню структуру сервісів. Nmap підходить для швидкого аналізу, але не забезпечує такої ж деталізації.



```
I Summary
=========

This document reports on the results of an automatic security scan.
The report first summarises the results found.
Then, for each host, the report describes every issue found.
Please consider the advice given in each description, in order to rectify
the issue.

All dates are displayed using the timezone "Coordinated Universal Time",
which is abbreviated "UTC".

Vendor security updates are not trusted.

Overrides are off.  Even when a result has an override, this report uses
the actual threat of the result.

Notes are included in the report.Information on overrides is included in the report.

This report might not show details of all issues that were found.
Issues with the threat level "High" are not shown.
Issues with the threat level "Medium" are not shown.
Issues with the threat level "Low" are not shown.
Issues with the threat level "Log" are not shown.
Issues with the threat level "Debug" are not shown.
Issues with the threat level "False Positive" are not shown.
Only results with a minimum QoD of 70 are shown.

This report contains results 1 to 10 of the 98 results selected by the
filtering described above.  Before filtering there were 419 results.

Scan started: Mon Oct 28 11:55:22 2024 UTC
Scan ended:   Mon Oct 28 13:10:58 2024 UTC
Task:         Task 192.168.3.234

Host Summary
************

Host           High  Medium  Low  Log  False Positive
192.168.3.234     2       2    0    6               0
Total: 1          2       2    0    6               0


II Results per Host
===================

Host 192.168.3.234
******************

Scanning of this host started at: Mon Oct 28 11:56:57 2024 UTC
Number of results: 10

Port Summary for Host 192.168.3.234
-----------------------------------

Service (Port)          Threat Level
80/tcp                  High (CVSS: 10.0)
631/tcp                 Log (CVSS: 0.0)
general/CPE-T           Log (CVSS: 0.0)
general/tcp             Log (CVSS: 0.0)

Security Issues for Host 192.168.3.234
--------------------------------------

Issue
~~~~~
NVT:    Apache HTTP Server Detection Consolidation
OID:    1.3.6.1.4.1.25623.1.0.117232
Threat: Log (CVSS: 0.0)
Port:   general/tcp

Summary:
Consolidation of Apache HTTP Server detections.

Vulnerability Detection Result:
Detected Apache HTTP Server
Version:      2.4.7
Location:     80/tcp
CPE:          cpe:/a:apache:http_server:2.4.7
Concluded from version/product identification result:
Server: Apache/2.4.7 (Ubuntu)

Solution:

Log Method:
Details:
Apache HTTP Server Detection Consolidation
(OID: 1.3.6.1.4.1.25623.1.0.117232)
Version used: 2024-03-08T15:37:10Z

References:
 url: https://httpd.apache.org


Issue
~~~~~
NVT:    Apple / OpenPrinting CUPS Detection (HTTP)
OID:    1.3.6.1.4.1.25623.1.0.900348
Threat: Log (CVSS: 0.0)
```

Рисунок 3.3 –Детальний опис сканування за допомогою OpenVAS

2. Безпека результатів:

Отримані результати зберігаються у безпечному форматі на Raspberry Pi і доступні лише через Telegram-бот, що зменшує ризик несанкціонованого доступу

до звітів. Використання бота для управління доступом до звітів забезпечує додатковий рівень безпеки.

3. Придатність для різних типів аналізу:

Nmap підходить для початкового сканування і збору базової інформації, тоді як OpenVAS доцільно використовувати для комплексного аналізу та виявлення критичних вразливостей.

ВИСНОВКИ

В ході роботи було розроблено систему для автоматизованого тестування на проникнення на основі Raspberry Pi, що включає віддалене управління через Telegram бот, забезпечення безпечного тунелювання через VPN та інтеграцію інструментів для сканування мереж, таких як Nmap та OpenVAS.

Актуальність даної теми зумовлена зростанням кількості кібератак на інформаційні ресурси та необхідністю забезпечення належного рівня захисту в умовах обмежених фінансових і технічних ресурсів. Використання доступних рішень, таких як Raspberry Pi для пентесту, дозволяє підвищити рівень безпеки, що є важливим для малого бізнесу та організацій.

Розроблена система забезпечує кілька рівнів захисту та управління процесом тестування на проникнення, що робить її гнучкою і безпечною. Telegram бот, що працює на проміжному сервері, виступає центральним вузлом для управління та моніторингу VPN-сесій, а також для виконання команд сканування мережі. Також було реалізовано механізм автоматичного завершення VPN-сесій для підвищення безпеки.

Для тестування системи було налаштовано вразливу машину, що імітує внутрішню мережу, де проводились сканування на вразливості. Інструменти Nmap і OpenVAS продемонстрували високу ефективність у виявленні потенційних вразливостей в мережевій інфраструктурі, що підтверджує необхідність використання таких систем для підвищення безпеки мереж.

Отримані результати показали, що запропонована система є ефективною, мобільною та економічно вигідною для виконання тестів на проникнення, що дозволяє її рекомендувати для використання в організаціях з обмеженими ресурсами для кібербезпеки. Подальші рекомендації включають розширення функціоналу Telegram бота, інтеграцію додаткових інструментів моніторингу, а також покращення механізмів аутентифікації для підвищення рівня безпеки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **2023 Data Breach Investigations Report** URL: https://www.verizon.com/business/resources/reports/dbir/2023/summary-of-findings/ (дата звернення: 16.11.2024).
2. **Addressing Illegal Download** URL: https://about.att.com/story/2024/addressing-illegal-download.html (дата звернення: 16.11.2024).
3. **The 2023 State of the Internet Report** URL: https://censys.com/the-2023-state-of-the-internet-report/ (дата звернення: 16.11.2024).
4. **Cybersecurity Attacks in the Era of Generative AI** URL: https://www.cfo.com/news/cybersecurity-attacks-generative-ai-security-ransom/692176/ (дата звернення: 16.11.2024).
5. **Rising Cloud Threats Demand Advanced Defenses** URL: https://www.checkpoint.com/press-releases/rising-cloud-threats-demand-advanced-defenses-check-points-2024-report-highlights-urgent-need-for-ai-and-prevention-first-security-measures/ (дата звернення: 16.11.2024).
6. **Sophos State of Ransomware 2024 Report** URL: https://assets.sophos.com/X24WTUEQ/at/9brgj5n44hqvgsp5f5bqcps/sophos-state-of-ransomware-2024-wp.pdf (дата звернення: 16.11.2024).
7. **Pwnagotchi Project** URL: https://pwnagotchi.ai (дата звернення: 16.11.2024).
8. **Pi-hole Documentation** URL: https://docs.pi-hole.net (дата звернення: 16.11.2024).
9. **RaspiWiFi Project URL**: https://github.com/jasbur/RaspiWiFi (дата звернення: 16.11.2024).
10. **Kamatera Cloud Solutions** URL: https://kamatera.com/ (дата звернення: 16.11.2024).
11. **OpenVPN Install Script** URL: https://github.com/angristan/openvpn-install (дата звернення: 16.11.2024).
12. **Raspberry Pi Software** URL: https://www.raspberrypi.org/software/ (дата звернення: 16.11.2024).
13. **Metasploitable3 for Ubuntu 14.04 Upgraded** URL: https://sourceforge.net/projects/metasploitable3-ub1404upgraded/ (дата звернення: 16.11.2024).