Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп'ютерних наук та штучного інтелекту

Кафедра кібербезпеки інформаційних систем, мереж і технологій

До захисту допущено
Кафедрою КІСМіТ протокол №____ від «___» грудня 2025 р.

завідувач кафедри _____ Марина ЄСІНА
(підпис) *(ім'я, прізвище)*
«___» грудня 2025 р.

Кваліфікаційна робота

здобувача другого (магістерського) рівня вищої освіти

Аналіз вразливостей інформаційних систем з використанням етичного хакінгу та тестування на проникнення

Спеціальність (спеціалізація) 125 «Кібербезпека та захист інформації»

Освітня програма «Безпека інформаційних і комунікаційних систем»

Виконавець _____ Нікіта ЛАПАНИК
(підпис) *(ім'я, прізвище)*

Науковий керівник _____ Ольга МЕЛКОЗЬОРОВА
(підпис) *(ім'я, прізвище)*

Харків – 2025

РЕФЕРАТ

Пояснювальна записка до проекту магістра містить 67 сторінок, 40 рисунків, 1 таблицю, 1 додаток, 25 посилань на джерела.

Мета роботи полягає в дослідженні методів аналізу інформаційних систем за допомогою етичного тестування на проникнення, основних інструментів та методів, які використовуються в тестуванні на проникнення, а також у аналізі та верифікації вразливостей.

Об'єктом дослідження є процес тестування інформаційних систем на предмет виявлення вразливостей та проникнення.

Предметом дослідження є інструменти та методи, які використовуються під час етичного тестування, вплив штучного інтелекту на процес тестування, етапи сканування аналізу та верифікації вразливостей.

Основними методами дослідження є аналіз найпоширеніших інструментів для етичного тестування, порівняння підходів та виконання верифікації вразливостей. У роботі досліджено явища вразливостей та етичного тестування на проникнення, етапи та методи тестування, ключові інструменти для сканування інформаційних систем та методи експлуатації знайдених вразливостей. Особливу увагу було приділено реалізації модульного застосунку для сканування систем із використанням штучного інтелекту.

Результати роботи можуть бути використані для покращення структурованого підходу до процесу тестування інформаційних систем та в різних наукових роботах. Підхід із використанням штучного інтелекту дозволить підвищити ефективність процесу тестування на проникнення та автоматизувати рутинність у скануванні.

Продовження досліджень можуть мати вектор повноцінної розробки унікального інструменту для сканування інформаційних систем із можливістю аналізувати вихідні дані за допомогою генеративного штучного інтелекту. Також, можливим напрямом розвитку досліджень може бути дослідження додаткових

інструментів для етичного тестування на проникнення та створення нових методологій та підходів, з урахуванням нових викликів та тенденцій.

Ключові слова: ІНФОРМАЦІЙНІ СИСТЕМИ, PENTESTING, PENETRATION TESTING, ЕТИЧНЕ ТЕСТУВАВННЯ НА ПРОНИКНЕННЯ, ВЕРИФІКАЦІЯ ВРАЗЛИВОСТЕЙ, ВРАЗЛИВОСТІ ІНФОРМАЦІЙНИХ СИСТЕМ, СКАНУВАННЯ СЕРВІСІВ, СКАНУВАННЯ ПРОТІВ, ЕТИЧНИЙ ХАКІНГ, ІНСТРУМЕНТИ ЕТИЧНОГО ХАКІНГУ.

ABSTRACT

The master project explanatory note contains 67 pages, 40 figures, 1 table, 1 appendix, and 25 references.

The purpose of the work is to study methods for analyzing information systems through ethical penetration testing, the main tools and techniques used in penetrations testing, as well analysis and verification of vulnerabilities.

The object of the research is the process of testing information systems for the detection of vulnerabilities and penetration.

The subject of the research includes the tools and methods used during ethical testing, the impact of artificial intelligence on the testing process, and the stages of scanning, analysis, and verification of vulnerabilities.

The main research methods are the analysis of the most common tools for ethical testing, comparison of approaches, and performing vulnerability verification.

The study examines the phenomena of vulnerabilities and ethical penetration testing, the stages and methods of testing, key tools for scanning information systems, and methods of exploiting detected vulnerabilities. Special attention was paid to the implementation of modular application for system scanning using artificial intelligence.

The results of the work can be used to improve a structured approach to the process of testing information systems and in various scientific studies. The approach involving artificial intelligence will increase the efficiency of penetration testing and automate routine scanning tasks.

Future research may focus on the full development of a unique tool for scanning information systems with the ability to analyze output data using generative artificial intelligence. Another potential direction is the study of additional tools for ethical penetration testing and the creation of new methodologies and approaches, taking into account emerging challenges and trends.

Keywords: INFROMATION SYSTEM, PENTESTING, PENETRATION TESTING, ETHICAL PENETRATION TESTING, VULNERABILITY VERIFICATION, INFROMATION SYSTEM VULNERABILITIES, SERVICE

SCANNING, PORT SCANNING, ETHICAL HACKING, ETHICAL HACKING TOOLS.

ЗМІСТ

ПЕРЕЛІК ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

CVE	- Common Vulnerabilities and Exposures
MITM	- Man in the Middle
CVSS	- Common Vulnerability Scoring System
ROE	- Rules of Engagement
ISO	- International Organization for Standardization
IEC	- International Electrotechnical Commission
NIST	- The National Institute of Standards and Technology
OWASP	- Open Web Application Security Project
OSSTM	- Open Source Security Testing Methodology Manual
PoC	- Proof of Concept

ВСТУП

Сучасні інформаційні системи, які активно використовують мережеві сервіси та веб-технології, набувають проблеми критичного значення - захист даних в умовах постійно зростаючих загроз та вразливостей. Разом з тим, відбувається і зростання складності програмних продуктів, використання сторонніх сервісів, компонентів та послуг, що призводить до збільшення інформаційної інфраструктури в рамках будь-якої організації. Усе це створює широку поверхню атаки, що робить такі системи привабливими цілями для різноманітних кіберзагроз. Через це, окрім стандартних процедур забезпечення безпеки, з'являється необхідність у превентивних заходах, які дозволять ідентифікувати слабкі місця системи до того, як ними скористається зловмисник.

Одним із найефективніших рішень є використання методів етичного хакінгу для аналізу вразливостей інформаційних систем. Це дозволяє імітувати дії потенційних злочинців, практично та безпечно відтворювати реальний вплив на інформаційну систему, верифікувати вразливості та реалізовувати звіти із практичними рекомендаціями щодо методів уникнення потенційних наслідків.

Метою роботи є дослідження інструментальної бази та практичних підходів до виявлення і верифікації вразливостей у веб-додатках та мережевих сервісах. Завдання роботи полягає у визначенні конкретних кроків, які є необхідними для визначення вразливостей, їх використання та аналізу їхнього впливу на інформаційну інфраструктуру.

В рамках додаткового дослідження, була поставлена ціль дослідити інтеграцію штучного інтелекту в процес етичного тестування на проникнення. Світові тренди призвели до того, що інформаційні системи наповнюються штучним інтелектом для автоматизації людської діяльності в цих самих системах. З такою метою можна дослідити наявні інструменти, які автоматизують процес етичного тестування на проникнення за допомогою модулів штучного інтелекту. Окремий інтерес в цій області може викликати генеративний штучний інтелект, який дозволить на основі наявних даних взаємодіяти із системою більш ефективно.

Результати роботи можуть бути спрямовані на застосування в галузях інформаційної безпеки, розробки політик управління вразливостями та підготовки навчальних лабораторій для визначення та верифікації вразливостей.

1 ТЕОРЕТИЧНІ ОСНОВИ АНАЛІЗУ ВРАЗЛИВОСТЕЙ ІНФОРМАЦІЙНИХ СИСТЕМ

1.1 Поняття інформаційних систем та їх загрози

Інформаційна система – це набір взаємопов’язаних компонентів, які інтегрують збір, обробку, зберігання і розподіл даних, інформації і цифрових продуктів для підтримки прийняття рішень, координації, контролю, аналізу та візуалізації в організації. Ці системи відіграють важливу роль в керуванні і спрощенні різних бізнес-процесів, а також можуть використовуватись у повсякденному житті. Це явище залежить від технологічних здобутків, пересікаючись з бізнесом, інформатикою та менеджментом, відіграючи важливу роль в потенційній ефективності, продуктивності і здібності конкурувати між організаціями. [10]

Через об’єм впливу інформаційних систем на більшу частину діяльності людства, їх розмір з плином часу почав сягати великих масштабів. Через це інформаційні системи стали розкладатися на інкапсульовані складники, або окремі компоненти, які утворюють зовнішню або внутрішню інфраструктуру. Серед цих компонентів можна виділити наступні:

- Апаратне забезпечення - це фізична частина до якої можна доторкнутись. Наприклад, сервер, комп’ютер, ноутбук, камера, сканер.
- Програмне забезпечення - це набір інструкцій, які виконуються апаратним забезпеченням. Можна виділити дві категорії програмного забезпечення: операційні системи та прикладні додатки. Операційні системи - це інтерфейс взаємодії між прикладним апаратним забезпеченням і обладнанням. Прикладами операційних систем можуть бути різні дистрибутиви Linux (Ubuntu, Kali, Debian), Microsoft Windows. Прикладне програмне забезпечення дозволяє користувачам взаємодіяти із операційними системами, виконувати як прості задачі, наприклад створення, видалення,

редагування файлів, так і взаємодіяти із внутрішнім ядром операційної системи для різних задач.

- Бази даних - це програмна реалізація для зберігання різних типів даних в різних форматах. Їх можна поділити на дві категорії, а саме: реляційні та нереляційні. Реляційні бази даних - це бази даних, які зберігають інформацію в таблиці, де вертикальний стовпчик вказує на різні властивості даних, а горизонтальна лінія є одним об’єктом, якому належать усі властивості описані в стовпчиках. Популярними прикладами реляційних баз даних є Postgresql, MySQL, Oracle. Нереляційні бази даних також зберігають в собі інформацію, але на відміну від табличного підходу, тут використовується вільний тип зберігання даних. Самі дані діляться в базі за логічним поділом і кожен об’єкт може містити різні властивості, які не задані для всієї бази даних.
- Комунікаційні мережі - це опціональна складова, яка дає можливість об’єднувати різні апаратні реалізації за допомогою, в тому числі, і програмного забезпечення за різними ознаками для комунікації пристроїв. Взагалі, інформаційна система може існувати і без можливості комунікувати, але реалії побудовані таким чином, що важко знайти готову, реалізовану інформаційну систему без побудованої під неї мережі.
- Користувачі - люди, які працюють з інформаційними системами, від спеціалістів служби підтримки до системних аналітиків, розробників та різних загальних категорій людей, які пов’язані із організацією. До загальної категорії відносяться усі, хто не обслуговує інформаційну систему, а користується нею для реалізації покладених обов’язків. Ця категорія може ділитися за різними властивостями: градація, цільова задача, посада та інше. Цей розподіл дозволяє масштабувати інформаційну систему для оптимізації роботи і для уникнення людського фактору, який відіграє важливу роль в контексті безпеки інформаційних систем і буде розглянутий пізніше.

На рисунку 1.1 зображена схема, яка демонструє основні компоненти інформаційної системи.

Рисунок 1.1 – Схема компонентів інформаційної системи

Як зазначалось раніше, інформаційні системи вже захопили більшу частину життєдіяльності людства. Прекрасним прикладом їх використання є електронний уряд. Це інформаційна система, концепція якої була створена на початку 90-их років, маючи на меті створити систему для отримання, надання, введення, збереження інформаційних ресурсів на вимогу користувача, для надання органами влади послуг фізичним та юридичним особам, а також інформувати їх про діяльність органів влади. В Україні ця концепція реалізована у додатку “Дія”, діяльність якого включає в себе велику інформаційну інфраструктуру.

Також, більшість державних структур включають в себе інформаційні системи, для ефективного виконання різного типу задач. Особливо важливим це питання стає в умовах війни. Сили оборони, як і більшість країни, переживає етап діджиталізації, все більше відходячи від старих методів комунікації.

Інформаційні системи використовуються і в бізнес рішеннях. Фінансові системи, надання послуг та загальна взаємодія із суспільством створює необхідність використання інформаційних систем і в повсякденному житті. Наприклад, соціальні мережі, онлайн-банкінг, хмарні середовища – це все інформаційні системи, які використовуються людиною, або користувачем та реалізовані в бізнесі. Всім цим інформаційним системам довіряються на обробку

дані, які самі по собі є доволі чутливими. Це означає, що будь яка, довірена одиниця даних, чи то логін та пароль, або документи чи особисті файли мають величезну цінність для користувачів, для яких є важливим збереження цих даних. Надійне збереження цих даних реалізується надійним захистом інформаційних систем від потенційних загроз.

Загалом, загрози інформаційних систем можна поділити на декілька категорій:

- Фізичні - це крадіжка обладнання, природні явища, наприклад пожежа чи повінь, фізичне пошкодження носіїв інформації.
- Організаційні - це загрози, які пов’язані із відсутністю політик безпеки, із людським фактором, або допущенням внутрішніх загроз.
- Технічні - ці загрози поділяються на 2 типи. Перший – це помилки конфігурації або налаштування. Цей аспект покладається на інженерів безпеки, системних адміністраторів та інженерів з інтеграції. Другий - це зовнішні загрози, які реалізують користувачі або потенційні кіберзлочинці. Це можуть бути ботнети, віруси, руткіти, експлуатація вразливого програмного забезпечення, DDos-атаки, несанкціонований доступ та інші.

1.2 Вразливості інформаційних систем

Як вже було з’ясовано, загроза інформаційної системи - це потенційно шкідливий суб’єкт, подія або обставина, які можуть використовувати вразливості для заподіяння шкоди або порушення роботи. В такому випадку виникає питання: “Що таке вразливість?”.

Вразливість інформаційної системи - це слабкі місця в технологіях, процесах або людях, які можна експлуатувати, при цьому вони являються загрозою тільки тоді, коли існує імовірність використати їх. Тож, якщо спростити визначення, можна сказати, що загроза - це потенційна небезпека, а вразливість - це “дірка”, через яку може реалізуватись загроза.

Кожен рік кількість вразливостей дуже стрімко зростає. За даними CVE в 2015 році загальна кількість задокументованих вразливостей склала майже 7,000, в

той час як станом на 2024 рік їх кількість складає трохи більше ніж 40,000. [6] В публікації ENSIA Threat Landscape 2024 зазначається, що в Європі з 1 липня 2023 року по 1 липня 2024 року, в NIST NVD було зареєстровано загалом 33,524 вразливості. Це показує значне збільшення порівняно з 24,690 вразливостями, про які повідомлялось в попередньому році між 1 липня 2022 року та 1 липня 2023 року. На рисунку 1.2 зображено графік із зростанням кількості задокументованих вразливостей з 2015 року по 2024 рік.

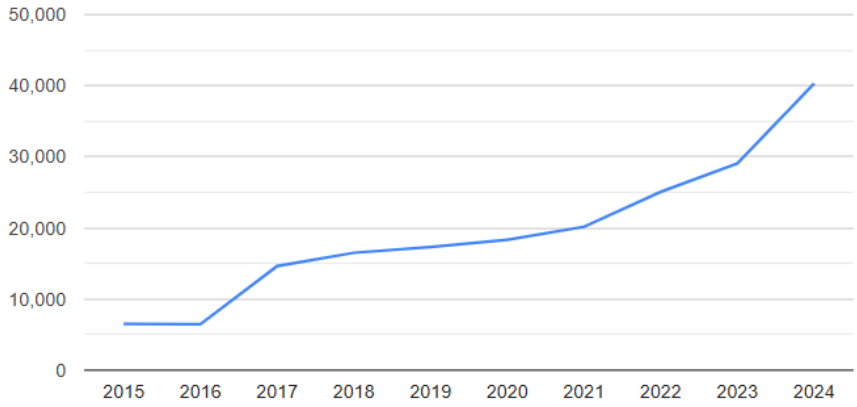


Рисунок 1.2 – Графік зростання кількості вразливостей

Існує багато підходів для класифікації вразливостей, але поділ за об’єктом є найефективнішим. В цьому підході система інкапсулюється таким чином, що кожний вразливий сегмент інформаційної системи має свої причини і наслідки. Класифікація вразливостей інформаційних систем за об’єктом включає у себе: апаратні, програмні, мережеві, процедурні та людські вразливості.

1.2.1 Апаратні вразливості

Слабкі місця або недоліки фізичних пристроїв, які можна використати для отримання доступу або спричинення шкоди. Наприклад, сервери, смартфони, ноутбуки, отримавши в руки обладнання, потенційний атакуючий може отримати доступ до критично важливих систем. Також, до апаратних вразливостей відносяться вразливості прошивки. Прошивка - це програмне забезпечення, яке працює на апаратному рівні. Вразливості прошивки можуть призводити до

постійних атак, адже їх доволі важко виявити. Виправлення таких вразливостей відбувається, частіше за все, під час оновлення самого програмного забезпечення.

1.2.2 Програмні вразливості

Помилки або недоліки в програмному забезпечені, які можуть бути експлуатовані для отримання доступу до системи, часто через помилки в коді або застарілі версії програмного забезпечення. Наприклад, одна з найпоширеніших проблем - відсутність встановлених правил оновлення безпеки додатків у системі. Розробники програмного забезпечення можуть з часом випускати оновлення з виправленням вразливостей. Без регулярного оновлення інформаційна система стає все більш вразливою. Разом із цим, навіть оновлені версії програмного забезпечення можуть мати небезпечний код. Неправильно написаний додаток або сервіс інтегрований до інформаційної системи може призвести до проблем, як SQL-ін’єкцій, або помилок в обробці вхідних даних, які в свою чергу призводять до витоку даних. Також до програмних вразливостей відноситься слабка аутентифікація. Доволі багато систем покладаються на слабкі методи аутентифікації, наприклад прості паролі або незашифровані форми вводу даних.

1.2.3 Мережеві вразливості

Слабкі місця в комп’ютерній мережі, наприклад відкриті порти, неправильно налаштовані брандмауери, незахищені безпровідні мережі які можуть бути використані потенційними атакуючими для перехвату конфіденційної інформації або отримання несанкціонованого доступу. Наприклад, непотрібні відкриті порти на пристрої можуть стати точкою входу атакуючого. Також відкриті або слабо захищенні, без надійного паролю і шифрування безпровідні мережі можуть дати доступ до інформаційної системи. Отримавши цей доступ, потенційний атакуючий може перехоплювати з’єднання, отримувати доступ до пристроїв, або атакувати підключені системи. Прикладом такої атаки можна назвати MITM або Людина посередині. Під час такої атаки, атакуючий перехоплює і змінює дані, які передаються між двома сторонами, роблячи видимість того, що дві сторони

взаємодіють один з одним, в той час як насправді дані передаються від пристроїв жертв до пристрою атакуючого.

1.2.4 Процедурні вразливості

Слабкі сторони процесів та правил, які встановлені в організації, наприклад використання паролів за замовчуванням або відсутність контролю дій співробітників, які можуть дозволити потенційним атакуючим обійти систему безпеки. Наприклад, багато пристроїв, додатків або систем надаються із конфігураціями за замовчуванням, які не мають оптимізації для забезпечення захисту. Коли організація залишає ці налаштування за замовченням, наприклад, стандартний пароль для входу в панель адміністратора, це надає потенційним атакуючим можливість легко проникнути в систему. Також, через неправильний контроль доступу, коли існує неправильне налаштування параметрів контролю доступу, наприклад надання конфіденційних даних неавторизованим користувачам, може виникнути загроза для системи. До процедурних загроз відноситься пряме ігнорування організацією журналювання та моніторинг. Ця проблема завадить своєчасно помітити несанкціонований доступ або іншу активність. Наприклад, відсутність моніторингу дозволить потенційному атакуючому мати постійний доступ до системи і залишатись непоміченим.

1.2.5 Людські вразливості

Ризики безпеки, викликані поведінкою людини, такі як слабкість перед фішинговими атаками, використання слабких паролів або слабка обізнаність в контексті загроз безпеці, що робить роботу потенційних атакуючих легшою. Це може досягатись різними шляхами. Загалом, поведінка людини доволі часто є найслабшим компонентом в кібербезпеці. Використання тактик соціальної інженерії змушує людей розкривати конфіденційну інформацію або чинити такі дії, які порушують встановлені правила безпеки. Також, деякі співробітники або користувачі можуть не спеціально створювати вразливості. Наприклад, використовуючи слабкі паролі, або передаючи облікові дані, або не блокуючи

пристрої, коли вони не використовуються, співробітники роблять себе легкою мішенню для потенційних атакуючих. Разом із тим, користувачі або співробітники можуть бути недостатньо обізнані в питаннях безпеки, що може зробити їх загрозою для інформаційної системи. Наприклад, не задумуючись переходити на шкідливі посилання або не розпізнати фішинговий лист.

1.2.6 Інші типи класифікацій вразливостей інформаційних систем

На сьогоднішній день, вразливість інформаційної системи є комплексним явищем. Попри те, що класифікація за об’єктом прояву є ефективним способом структурувати це явище, з точки зору практичної кібербезпеки існує значно більший перелік критеріїв. Це означає, що жодна класифікація не може бути всеохоплюючою.

На ряду із класифікацією за об’єктом прояву, поширеною є класифікація за походженням. Вона дозволяє зрозуміти, на якому етапі роботи інформаційної системи була допущена помилка. Наприклад, тут виділяється помилка при проектуванні системи. Це помилки, які закладаються у фундаментальні рішення при розробці та побудові інформаційної системи. Також є вразливості, які виникають при розробці програмного забезпечення та їх інтеграції в інформаційну систему. Наприклад, синтаксичні або логічні помилки. Доволі поширеними видами таких вразливостей є SQL-ін’єкції або неправильна робота з пам’яттю. Існує вразливість конфігураційного типу. Це недолік, який пов’язаний із недбалістю або помилкою людини під час налаштування систем, сервісів, обладнання або політик доступу.

Також, існує поділ за критичністю вразливості. Цей підхід використовується для того, щоб визначити пріоритети усунення вразливостей. Поширеною системою для такої класифікації є CVSS, яка присвоює бал критичності для кожної знайомої вразливості. Це допомагає поділити вразливості, відповідно до балу на критичні, високі, середні та низькі. Використовуючи таку класифікацію, можна визначити необхідні ресурси для того, аби усунути вразливість та забезпечити безпеку інформаційної системи.

Окрема класифікація визначається за впливом на ключові властивості безпеки інформаційних систем та їх змісту: конфіденційність, цілісність та доступність. Такий підхід дозволяє розуміти, які можливі наслідки буде мати успішна атака на систему. Наприклад, безпека доступності пов’язана із вразливостями мережевого характеру, тоді як цілісність даних знаходиться під загрозою логічних помилок. Конфіденційність пов’язана із людським фактором, описаним раніше.

Вказані підходи класифікації вразливостей інформаційних систем не суперечать із класифікацією за об’єктом. Навпаки, використання різних підходів дозволяє розширити уявлення про потенційні ризики і будувати більш надійні та ефективні процеси захисту інформації.

Розуміючи загальні принципи класифікації вразливостей інформаційних систем, доволі просто зробити висновок про те, які саме джерела виникнення вразливостей існують. Основним джерелом є розмір інформаційної системи. Чим більша система, тим більше шансів того, що вразливість з’явиться. При чому, розміри інформаційних систем зростають не тільки внутрішньо, а й зовнішньо. Реалізація взаємодії з іншими системами, наприклад для аутентифікації, авторизації, платіжні системи, підключення різних API, особливо актуальним стає із зростанням популярності штучного інтелекту, все це є зовнішнім зростанням інформаційних систем, що в перспективі створює потенційні вразливості.

Вразливості можуть бути не виявлені на початку роботи сервісів, а лише з плином часу. Через неправильну конфігурацію або ігнорування необхідності постійного оновлення програмного забезпечення, в інформаційній системі з’являються місця, які через деякий час можуть стати точкою входу або місцем експлуатації цієї системи. Разом із тим, навіть якщо система не розширюється і розроблена таким чином, що залишається доволі малою за обсягом технічних та програмних засобів, завжди існують помилки в розробці. Відсутність рев’ю, слабка обізнаність розробників у кібербезпеці та постійне перекладання відповідальності за безпеку на фреймворки та бібліотеки створює потенційні “діри” в захисті даних.

Різноманітність джерел виникнення вразливостей та їх задокументована кількість вказує на необхідність їх систематичного виявлення та аналізу. Одним із найефективніших підходів до цього є проведення етичного хакінгу та тестування на проникнення, які дозволяють змоделювати дії зловмисників і оцінити реальний рівень ризиків.

1.3 Принципи етичного хакінгу

Загалом, історія такого явища як “етичний хакінг” показує, що бути умовним, потенційним зловмисником - не завжди погано. Поняття “етичний хакінг” в сучасному контексті з’явилось в Масачусетському технологічному інституті. В 1995 році віце-президент IBM Джон Патрик вперше використав термін “етичний хакінг”, але сама концепція існувала на багато раніше. В 1960-их роках цей термін використовувався студентами інженерних факультетів, який означав пошук оптимізації систем і машин для підвищення їхньої ефективності. Тож, на той час це поняття мало характер творчої діяльності в області оптимізації. В 1990-их роках, коли інтернет почав набувати своєї популярності по всьому світу, кількість хакерів виросла. Також в цей час персональні комп’ютери почали набувати своєї популярності, що означало, що велика частина особистої інформації та інших конфіденційних даних тепер зберігається у вигляді комп’ютерних програм. Це підштовхнуло хакерів до спроб отримати доступ до цих систем для того, щоб продати отриману інформацію. На фоні цього засоби масової інформації почали поширювати проблему в негативному ключі. Хакерів почали сприймати як злочинців, які використовують свої навички для отримання доступу до приватних комп’ютерів, крадіжок даних та шантажу організацій. З цього моменту хакерів-злочинців почали називати “чорними капелюхами”. [8]

Разом із тим, люди з навичками притаманними для хакерів отримали ще декілька категорій, а саме “сірі капелюхи” та “білі капелюхи”. Як і злочинці, сірі капелюхи не мають дозволу на тестування та проникнення в інформаційну систему. Однак, на відміну від чорних капелюхів, це люди, які використовують свої навички на благо інших, іноді ігноруючи закони та етичні норми. Наприклад, виявивши

шахрайський сайт, така людина цілеспрямовано буде атакувати всю інформаційну систему, нехтуючи законами, для того, щоб люди в мережі не постраждали від використання подібного ресурсу.

В свою чергу, “білі капелюхи” або тестери - це хакери, які дотримуються закону та використовують свої навички на благо інших, думаючи про етичність своїх дій. Наприклад, тестер на проникнення виконує авторизоване тестування в компанії. В цілому, поняття законності та етики в кібербезпеці доволі суперечливі. Тест на проникнення - це авторизований аудит безпеки для виявлення ризиків і загроз в інформаційній системі, узгоджений з власником цієї системи.

Перед тим, як починається тест на проникнення, відбувається офіційне обговорення між тестером та власником. Узгоджуються методи, які будуть використані, системи, які можна і потрібно тестувати, інструменти, які дозволено використовувати, щоб в процесі тестування система випадково чи спеціально не була пошкоджена. Цей процес вирішує питання законності такої діяльності, але питання етики залишається відкритим, навіть після офіційного дозволу.

В даному контексті, етика - це моральна дискусія між правильними і неправильними діями. Дія може бути законною, але водночас суперечити системі переконань людини щодо правильного і неправильного. Тестери на проникнення часто стикаються із рішеннями, які потенційно можуть бути морально сумнівними. Наприклад, етичний хакер отримав доступ до баз даних, в яких можуть зберігатись конфіденційні, особисті або чутливі дані. Або може виникнути ситуація, коли тестеру потрібно провести фішингову атаку на співробітника, для перевірки безпеки персоналу компанії або організації. Законність такої дії може бути узгодженою, але залишатись етично сумнівною.

Разом із тим, перед початком тестування на проникнення створюється документ – ROE. Цей документ має три основні розділи, які відповідають за всі рішення щодо проведення тестування.

Перший розділ – дозволи. Цей розділ надає чіткі дозволи на виконання тестування. Він є важливим для юридичного захисту фізичних осіб та організацій за діяльність, яку вони здійснюють.

Другий розділ - область тестування або, як частіше кажуть в сфері – scope. У цьому розділі документа зазначаються конкретні цілі, до яких має застосовуватись тестування, а також зазначаються цілі, які категорично заборонено тестувати. Наприклад, методи тестування дозволено застосовувати до певних серверів та додатків компанії, але не для всієї мережі. Це робиться для того, щоб функціонування інформаційної системи, в разі будь-яких проблем, які можуть виникнути в процесі тестування, не зупинилось.

Останній важливий розділ цього документу – це правила. У цьому розділі будуть точно визначені методи, які дозволені під час тестування та взаємодії із інформаційною системою. Наприклад, в правилах можуть бути заборонені методи фішингових атак, але атаки на облікові дані співробітників, або атаки, які імітують діяльність співробітників дозволені.

Виходячи з цього, прийнято розділяти тестування інформаційних систем на три типи. Розуміння цільової системи, сервісів та супутніх властивостей допомагає визначити рівень тестування, який буде необхідним під час аналізу системи.

1.3.1 Тестування чорної скриньки (Black-Box)

Це високорівневий процес тестування, під час якого тестувальник не має жодної інформації про внутрішню роботу програми, сервісу, мережі чи інформаційної системи. Тестувальник діє як звичайний користувач, який тестує взаємодію програми із іншими програмами в системі чи поза нею, функціонал продукту та його реалізацію. Таке тестування може включати в себе взаємодію із інтерфейсом, кнопками, полями вводу, перевірку того, чи повертається бажаний результат. При тестуванні чорної скриньки немає потреби в знаннях програмування, або розумінні того, як працює програма. Це, скоріше, пошук вразливостей, які виникли на етапі розробки, або при додаванні до інформаційної системи нової компоненти, але все ще залишаються небезпечними для системи. При тестуванні методом чорної скриньки сильно збільшується час на фазі збору інформації, для того, щоб знайти цілі для атаки, або потенційні точки входу.

Важливою рисою такого тестування є те, що існує можливість виявляти логічні вразливості, які не є очевидними для розробників. Тесувальник не має обмежень реалізації внутрішнього продукту і взаємодіє з системою так, наче це зробив сторонній користувач, і може виявити помилки логіки, некоректні обробки даних, або неправильну роботу сервісів.

Умови такого тестування – одні з найважчих у плані часу та використання ресурсів, адже є необхідність у зборі великого обсягу даних, від простої розвідки у відкритих джерелах до повного мережевого сканування. Через це, тестування чорної скриньки, під час тестування є точкою старту, щоб отримати реальне уявлення про те, як саме система виглядає для сторонніх.

1.3.2 Тестування сірої скриньки (Grey-Box)

Це найпопулярніший процес тестування інформаційних систем на проникнення. Він має поєднані процеси тестування як “чорної скриньки”, так і “білої скриньки”. Тестер має обмежені знання про функціонування інформаційної системи та про внутрішні компоненти програм або програмного забезпечення. При цьому, при тестуванні буде відбуватись взаємодія із інформаційною системою так, ніби відбувається сценарій “чорної скриньки”, а потім отримані знання про програму або систему використовуються для того, щоб спробувати вирішити проблеми в міру їх виявлення.

При такому типі тестування обмежені знання про функціонування програми або системи дозволяють економити час і тестувати більш глибші межі. Це і є унікальною перевагою сірої скриньки, адже тестувальник знає який стан системи є оптимальним при звичайному режимі роботи. Підхід дозволяє протестувати реальні сценарії підвищення привілеїв у системі. Наприклад, якщо атакуючий має доступ до внутрішніх сервісів, він може моделювати поведінку внутрішнього користувача, який шукає можливість розширити свої права чи отримати доступ до чутливої інформації.

Також унікальність такого типу тестування полягає ще й у можливості знаходити проблемні місця в інформаційній системі до того, як відбудеться етап

розгортання, не витрачаючи зайвий час, ресурси та не ризикуючи чутливими даними. Особливо корисним, такий підхід може бути у системах, в яких існує велика кількість мікросервісів, адже балансує глибину перевірки.

1.3.3 Тестування білої скриньки (White-Box)

Цей процес тестування та аналізу є низькорівневим, який зазвичай виконується розробником програмного забезпечення, який знає функціонал та логіку застосунку. Під час тестування білої скриньки тестуються внутрішні компоненти застосунку або програмного забезпечення, які є частиною інформаційної системи. Маючи повне уявлення про програму та її очікувану поведінку, цей процес буде займати набагато більше часу, ніж тестування методом “чорної скриньки”.

Також повні знання про очікувану поведінку системи, можуть гарантувати, що буде охоплено найбільша можлива поверхня атаки. Це дозволяє провести різноманітні тестування з різними сценаріями функціонування і реакції різних частин інформаційної системи.

Через те, що тестувальник має доступ до всієї інформації, подібний тип тестування є цінним для критичних систем – банківські або промислові інформаційні системи та системи зі складними правилами доступу.

Кожен із цих типів тестування має технічні особливості та методи, включаючи різні стилі взаємодії, різну глибину аналізу та різні масштаби перевірки. Визначення різних тестувань робить сучасний пентестинг не просто перевіркою на наявність вразливостей, а аналітичним комплексним процесом, який має бути адаптованим під конкретні вимоги архітектури, специфіки завдань та загроз конкретної інформаційної системи.

Для впровадження аналізу інформаційних систем методами етичного хакінгу використовуються методики, стандарти та нормативна база, які регламентують цей процес.

1.4 Стандарти та нормативна база в кібербезпеці

Стандартизація в кібербезпеці - це узагальнення правил того, як саме має будуватись система захисту інформації. В ці правила входить і аналіз та тестування інформаційних систем, як частина процесу захисту інформації. Більшість країн світу приймають закони, які вимагають від компаній реалізовувати такі правила, якщо компанія надає послуги будь якого характеру і, особливо, якщо взаємодіє з чутливими даними своїх клієнтів. Стандартизація, як явище поділяється на три види: міжнародна стандартизація – стандартизація, яка проводиться на міжнародному рівні та участь у якій відкрита для відповідних органів усіх країн; регіональна стандартизація – стандартизація, яка проводиться на відповідному рівні країн повного географічного та економічного простору; національна стандартизація – стандартизація, яка проводиться на рівні однієї країни.

Саме міжнародні стандарти будують основу для глобальної практики тестування на проникнення. Вони фокусуються на методологіях, етичних принципах та управлінні ризиками. Серед таких стандартів найпоширенішими є:

ISO/IES 27001 – стандарт, розроблений Міжнародною організацією із стандартизації та Міжнародною електротехнічною комісією, є фундаментальним для систем управління інформаційною безпекою.[9] Сюди включені вимоги до проведення аудитів, тестування на проникнення та ідентифікації вразливостей як частина регулярної оцінки ризиків. У рамках цього документу етичний хакінг повинен бути документально оформленим, із чітко визначеними межами тестування та правилами. Також, в стандарті сформовані конфіденційність даних та відповідальність за можливі пошкодження систем.

NIST SP 800-115 – документ, розроблений Національним інститутом стандартів і технологій США, надає детальну методологію для тестування на проникнення.[23] Тут описуються фази аналізу інформаційних систем. Стандарт акцентує увагу на етичних аспектах процесу тестування, особливо на отриманні дозволу від власника інформаційної системи та уникненні реальних пошкоджень. Документ широко використовується в урядових та корпоративних середовищах.

OSSTMM – відкрита методологія, яка була розроблена Інститутом безпеки та відкритих методологій і фокусується на об’єктивному тестуванні безпеки. Вона охоплює різні канали атаки, наприклад, мережеві, або фізичні, або бездротові та забезпечує вимірюваність результатів.[18] Ця методологія підходить для аналізу інформаційних систем методами етичного хакінгу, оскільки наголошує на верифікації вразливостей без експлуатації.

В Україні регулювання етичного хакінгу інтегровано в загальну систему кібербезпеки, з урахуванням міжнародних стандартів. Закон України “Про основні засади забезпечення кібербезпеки України” визначає етичний хакінг як інструмент для оцінки критичної інформаційної інфраструктури. У 15 статті цього закону регулюється безпека проведення тестування на проникнення та необхідність отримання дозволу від Державної служби спеціального зв’язку та захисту інформації України. Закон вимагає фіксації результатів та повідомлення про виявленні вразливості. Подібні закони, а також розробка стратегій кібербезпеки України підкреслюють роль етичного хакінгу в захисті інформаційних систем, з акцентом на навчання фахівців та сертифікації. Також, в Україні діють рекомендації Комп’ютерної служби реагування на надзвичайні події, щодо тестування інформаційних систем, які базуються на OWASP та NIST.

Стандарти та нормативна база забезпечують ефективність та легітимність етичного хакінгу. Це процес контролю, який перетворює аналіз інформаційних систем з потенційно ризикованої діяльності на структурований процес. Дотримання цих норм, при роботі з інформаційними системами, дозволяє уникнути юридичних ризиків та забезпечити підвищення безпеки.

ВИСНОВОК ДО РОЗДІЛУ 1

Використання методів тестування на проникнення є надзвичайно важливою складовою в процесі захисту інформації та формуванні безпечного каркасу для зберігання даних в інформаційних системах. Це є актуальним для більшості існуючих організацій та структур, незалежно від того, якого роду послуги вони надають або, які завдання вирішують.

Було розглянуто поняття інформаційних систем та вразливостей, досліджені принципи етичного хакінгу та нормативна складова, разом із стандартами, які регулюють аналіз інформаційних систем. Проблемою популярних стандартів та загальних принципів є відсутність уніфікованого інструментарію для аналізу інформаційних систем.

Спираючись на це, створення власного підходу та методу аналізу інформаційних систем, але в рамках популярних стандартів, є важливим кроком для захисту інформації та забезпечення безпеки даних. Це може бути використано на різних етапах тестування на проникнення та аналізу інформаційних систем і бути спрямоване на підвищення ефективності процесу етичного хакінгу.

2 МЕТОДИ І ЗАСОБИ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ

2.1 Етапи тестування інформаційних систем

Тестування на проникнення – це процес, під час якого симулюється атака на інформаційну систему, щоб знайти та виправити будь-які наявні вразливості, перш ніж справжні зловмисники зможуть їх використати. На відміну від дій справжніх зловмисників, дії під час тестування на проникнення, зазвичай не мають єдиної, чіткої мети. Задача полягає в тому, що потрібно досліджувати систему повністю, а не шукати конкретну точку входу або можливість експлуатації. Для того, щоб процес аналізу не був хаотичним виділяють етапи тестування на проникнення:

- Підготовка та планування – етап включає визначення обсягу та цілей тестування на проникнення. При аналізі доволі важливо зрозуміти, які системи, підсистеми, мережі, додатки будуть протестовані та які першочергові методи будуть застосовані. Також, під час цього етапу вивчаються правила та вимоги до тестування.
- Розвідка – етап під час якого збирається якомога більша кількість інформації про ціль або організацію. Це може включати в себе такі деталі, як імена мереж, записи доменів, електронні пошти, номера телефонів та будь-яка загальнодоступна інформація про інфраструктуру організації. Поширеною практикою на цьому етапі є використання методологій OSINT.
- Сканування та аналіз – етап включає в себе виявлення програм та служб, що працюють у інформаційній системі. Це досягається за допомогою різних інструментів та методів, які допомагають зрозуміти, як цільова система реагує на спроби вторгнення. Сканування включає перевірку коду без його виконання. Це допомагає виявити вразливості на основі логіки та структури програмного забезпечення, яке працює в системі. Також, це може бути пошук сервісів, які включені в інформаційну систему, або які є її частиною. Наприклад, пошук веб-сервера, який може бути потенційно вразливим.
- Експлуатація – етап включає використання вразливостей, які були виявлені у системі. Процес використання вразливостей може включати в себе

використання публічних експлойтів, які були виявлені раніше та мають алгоритм дій, щодо того, як саме їх використовувати. Публічні експлойти - це загальновідомі вразливості, які поширюються в публічних базах знань спеціальними організаціями, або звичайними зацікавленими особами, які допомагають поповнювати такі бази. Також, у використанні вразливостей, окрім публічних експлойтів, включається експлуатація логіки програми для отримання несанкціонованого доступу до цільової системи. Використовуючи логіку програми, до процесу тестування не залучаються сторонні інструменти чи програми. У поширені вразливості такого типу входять: SQL-ін'єкції, міжсайтовий скриптинг та інші. Доволі часто, в етап експлуатації включають ескалацію привілеїв. Після успішного використання вразливостей системи, цей етап стає спробою розширити доступ до системи. Існує два варіанти ескалації привілеїв: горизонтально та вертикально. Горизонтальна ескалація привілеїв - це спроба отримати доступ до іншого облікового запису з тією ж групою дозволів. Наприклад, отримати доступ до облікового запису іншого користувача. Вертикальна ескалація привілеїв - це спосіб отримати доступ до іншої групи дозволів. Наприклад, до облікового запису адміністратора.

- Пост-експлуатація – етап містить в собі декілька вкладених етапів. По-перше, на цьому етапі відбувається пошук інших доступних хостів, або кінцевих точок, які можуть бути цільовими. По-друге, відбувається пошук додаткової інформації, яку можна отримати від хоста. Це особливо важливо, адже будучи привілейованим користувачем, атакуючий може знайти будь-що на цільовій системі, що створює додаткові загрози. По-третє, на цьому етапі замітаються сліди присутності атакуючого в системі. Це потрібно для того, щоб за допомогою методів цифрової криміналістики було важче знайти атакуючого, його діяльність та точки входу. Також, це важливий етап для підтримки доступу. Однією із цілей зловмисників може бути закріплення в системі протягом тривалого періоду без подальшого виявлення. Замітання

слідів допомагає імітувати поведінку справжніх зловмисників і протестувати реакцію системи на подібні дії.

- Звітність – після того, як процес тестування системи завершується, результати об’єднуються в детальний звіт. В цьому звіті міститься інформація про успішно використані вразливості та їх опис, вплив на систему, критичність вразливостей, конфіденційні дані, до яких було отримано доступ, тестові облікові дані, тривалість невиявленого входу та, опціонально, поради з виправлення виявлених вразливостей.

Через різність інформаційних систем, загальні етапи тестування інформаційних систем не здатні забезпечити вичерпний список вимог до процесу тестування. Етапи, залишаючись в такому ж вигляді, мають різні наміри, залежачи від системи, її вимог, правил та цільової задачі. Підбір правильних етапів залежить від методологій тестування на проникнення.

2.2 Методології тестування інформаційних систем

Аналіз та тестування інформаційних систем можуть мати широкий спектр завдань і різний рівень складності цілей. З цієї причини жоден процес тестування на проникнення не є однаковим і не існує універсального підходу до його проведення.

Кроки, які виконуються при тестуванні на проникнення, називаються методологією. Практична методологія - це перелік різноманітних процесів, в якому виконані кроки відповідають ситуації. Наприклад, використовування кроків, які описані в методології для перевірки та аналізу безпеки веб-застосунку не є ефективними для аналізу та тестування безпеки мережі.

2.2.1 OSSTMM

The Open Source Security Testing Methodology Manual містить основу стратегій тестування інформаційних систем, програмного забезпечення, додатків, комунікацій та людського аспекту кібербезпеки. Методологія охоплює всі необхідні області для забезпечення інформаційної безпеки, використовуючи

стандартизовані процедури послідовного проведення тестів на проникнення. Структурно вона дуже деталізована та, як правило, використовує унікальні визначення. В процес тестування даною методологією включений науковий підхід. Тобто, тестові випадки вимірювані, повторювані та перевірені. В загальному плані, методологія знаходиться за межами інформаційних систем. Тут охоплюється соціальна інженерія, цифрова безпека, операційна та фізична безпека. При цьому, в більшій мірі, OSSTMM не прив'язана до конкретного інструментарію чи набору технологій, тобто орієнтована на спільноту.[18]

2.2.2 OWASP

Open Web Application Security Project - це фреймворк, що був розроблений спільнотою та має постійні оновлення, що допомагає будувати вимоги для тестування безпеки веб-застосунків та сервісів. Оновлення в цій методології, це публікація загальнодоступних звітів, в яких зазначаються десять поширеніших веб-вразливостей. Не дивлячись на те, що це доволі поширена методологія, OWASP не має жодної акредитації та не надає рекомендацій щодо конкретних життєвих циклів розробки програмного забезпечення. Але, разом із тим, ця методологія охоплює всі етапи тестування: від самого тестування до звітності та виправлення вразливих ділянок інформаційної системи. Одним з найважливіших елементів OWASP є Web Security Testing Guide.[19]

WSTG - це набір структурованих тестових випадків, слідуючи за якими, можна проводити тестування на проникнення. Структурно WSTG поділяється на 12 категорій, які повністю охоплюють спектр тестування веб-додатків. До цих категорій відносяться:

1) Збір інформації
2) Управління розгортанням
3) Авторизація
4) Вхідні дані
5) Автентифікація
6) Сеанси

7) Обробка помилок
8) Логіка
9) Криптографічне тестування
10) API
11) Сторона клієнта
12) Управління ідентифікацією

2.2.3 PTES

Penetration Testing Execution Standard - це методологія, яка пропонує чітку, поетапну, структуровану модель для тестування на проникнення, орієнтовану на реалістичне відтворення дій зловмисника з подальшою оцінкою безпеки системи. В методології пропонуються всі етапи, від першої взаємодії до оцінки та аналізу загроз після завершення. Це забезпечує врахування і бізнес-логіки, і аспектів технічного плану, гарантуючи, що під час тестування будуть враховані елементи інформаційної системи, які мають найбільше значення та місце в якому знаходяться вразливості. До ключових фаз тестування на проникнення за методологією PTES відносяться наступні:

1) Розвідка та збір даних
2) Моделювання загрози
3) Аналіз інформаційної системи та її вразливостей
4) Експлуатація
5) Звітність

2.2.4 NIST

The National Institute of Standards and Technology – це структура, яка використовується для покращення стандартів кібербезпеки та управління ризиками кіберзагроз. В методології викладені рекомендації для контролю безпеки інформаційних систем, як для критичної інфраструктури, так і для комерційних організацій. В переліку цих рекомендацій існує розділ із рекомендаціями щодо методології, яку потрібно використовувати при тестуванні на проникнення.

Методологія доволі часто оновлюється, тож існує багато варіацій фреймворку. Для тестування на проникнення ця структура, будучи прийнятою, як стандарт має велике значення. Жоден тест не може бути проведений без формального плану, узгодженого відповідно до методології. Результати тестування можуть бути інтегровані у програми управління ризиками, аудити чи звітності. При цьому, якщо тестування на проникнення проводяться на декількох сегментах інформаційної системи, методологія має рекомендації, які забезпечують єдині правила, що дозволяє об’єктивно порівнювати результати.[23]

Не дивлячись на те, що кожна методологія може існувати незалежно, в контексті етичного хакінгу вони не є ізольованими. Практики, які впроваджуються в спільноті, поєднують кілька підходів використовуючи в конкретний момент той, який найбільш доречний. Комбінування дає можливість охопити більший обсяг життєвого циклу тестування інформаційних систем. У таблиці 2.1 буде розглянуто порівняльний аналіз методологій, згідно з загальними етапами тестування на проникнення.

Таблиця 2.1 – Порівняння етапів тестування в контексті різних методологій

	PTES	OSSTMM	NIST	OWASP
Підготовка	Визначення цілей, обсягів, правил	Визначення політик тесту, правових аспектів	Визначення ресурсів, обсягу, цілей	Визначення тестової політики та планування
Розвідка	Пасивна та активна розвідка	Збір зовнішньої та внутрішньої інформації	Збір загальнодоступних та технічних даних	Збір інформації про ціль методами OSINT

Продовження таблиці 2.1

Сканування	Виявлення потенційних векторів атаки	Детальний збір інформації про служби, протоколи, користувачів	Виявлення відкритих портів, служб, конфігурацій	Виявлення конфігурацій, методів розгортання
Експлуатація	Використання вразливостей для отримання доступу	Моделювання атак відповідно до зібраних даних	Експлуатація знайдених вразливостей для досягнення цілей	Активне тестування за списком OWASP(XSS, auth bypass, тощо)
Пост-експлуатація	Закріплення, збір даних, визначення впливу	Підтвердження масштабів впливу, глибини компрометації	Оцінка впливу, збір доказів, документування	Перевірка наслідків, збір логів, очищення після тестування
Звітність	Створення технічного виконавчого звіту	Надання результатів і рекомендацій	Документування виявленого, рекомендації з покращення безпеки	Оформлення результатів тестування

Методології тестування на проникнення – це не просто набори правил. Це інструменти, за допомогою яких можна управляти процесом тестування на проникнення, які визначають його якість, легітимність та цінність для організацій. Завдяки методологіям тестування перетворюється із хаотичної технічної практики на один з найвпливовіших компонентів захисту інформаційної інфраструктури, який підвищує стійкість до реальних кіберзагроз.

2.3 Інструменти тестування на проникнення

Різноманітність інформаційних систем, методологій тестування та стандартів у інформаційній безпеці створюють необхідність у вивченні різноманітного

спектру інструментарію для процесу тестування на проникнення. Вважається, що будь який вид злому систем – це творчій процес, який вимагає технічних знань та вмінь писати та розробляти код. Це необхідність, яка циклічно створює виклики для індустрії. Злочинець розробляє програмне забезпечення для проведення атаки. Після успішного використання, інженери з інформаційної безпеки, оцінивши наслідки, мають створити подібний інструмент для подальшого тестування своїх систем і підготовки до нових атак. З цієї причини злочинець має створювати новий кастомний інструмент для того, щоб обійти оновлений захист організації.

Разом з тим, спільнота фахівців з кібербезпеки створює спільні бази даних, в яких люди можуть надавати доступ до коду програмного забезпечення, або саме програмне забезпечення для тестування інформаційних систем. Безліч інструментарію з подібних платформ стали неоговореним стандартом, або зручним помічником в процесі аналізу. Для успішної інтеграції, такі інструменти мають бути, або з відкритим вихідним кодом, або випущені сертифікованими організаціями. Серед найпоширеніших інструментів аналізу та етичного тестування інформаційних систем виділяються: Nmap, Nessus та Metasploit Framework.

2.3.1 Nmap

Nmap (Network Mapper) – це безкоштовне програмне забезпечення з відкритим кодом, яке було створене Гордоном Лайоном в 1997 році. Інструмент використовується для картографування мереж, ідентифікації активних хостів та виявлення запущених служб. Також існує вбудований скриптовий механізм, який розширює функціональність та додає можливість отримувати відбитки роботи служб, або використовувати вразливості. Nmap проектувався як інструмент для реалістичної і масштабної карти мережі. Архітектура оптимізована під паралельні сканування, асинхронні тайм-аути та розширення через модульну систему. Завдяки такому дизайну він ефективно працює як на одиночних хостах, так і при обстеженні великих інформаційних систем з великими адресними просторами, маючи можливість контролювати навантаження на джерело сканування.[16]

Основний внесок у процес тестування на проникнення, Nmap вносить на етапі розвідки та сканування. В цьому контексті головною задачею сканеру стає перевірка того, які порти відкриті та прослуховуються, а які порти закриті. Використовуються три типи сканування портів: сканування портів TCP-з'єднанням, сканування TCP SYN-портів та сканування UDP-портів.

Сканування TCP-з'єднанням працює шляхом завершення тристороннього рукостискання. Під час встановлення стандартного TCP-з'єднання клієнт надсилає TCP-пакет із встановленим прапорцем SYN, а сервер відповідає SYN/ACK, якщо порт відкритий. Далі клієнт завершує тристороннє рукостискання, надсилаючи ACK.[4] На рисунку 2.1 зображена схема тристороннього рукостискання.

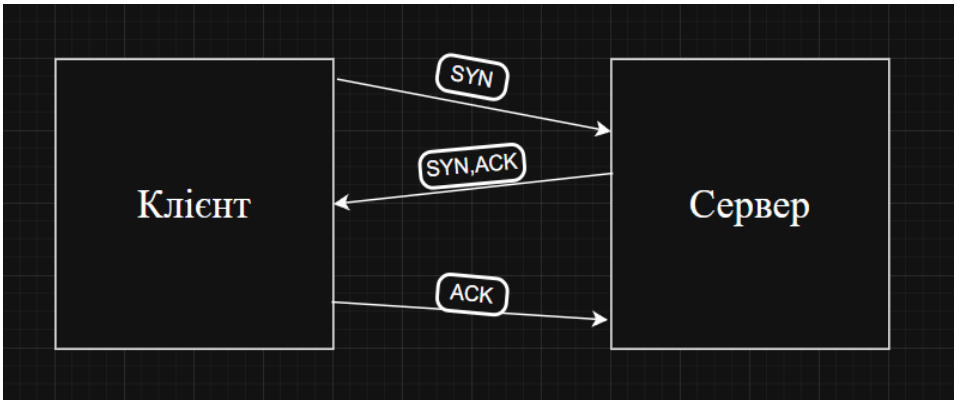


Рисунок 2.1 – Схема тристороннього рукостискання

При такому скануванні TCP-з'єднанням, задача полягає у тому, щоб з'ясувати чи відкритий TCP-порт, а не встановлення з'єднання з ним. Тому з'єднання розривається, коли стан порту підтверджується надсиланням RST/ACK. Сканування запускається за допомогою флагу "-sT". Повна команда: "nmap -sT target_ip". Схема при якій розривається з'єднання зображена на рисунку 2.2. Результат виконання сканування TCP-з'єднанням зображений на рисунку 2.3. Сканування було виконано у локальному середовищі, в якому запущені сервіси TCP та UDP.

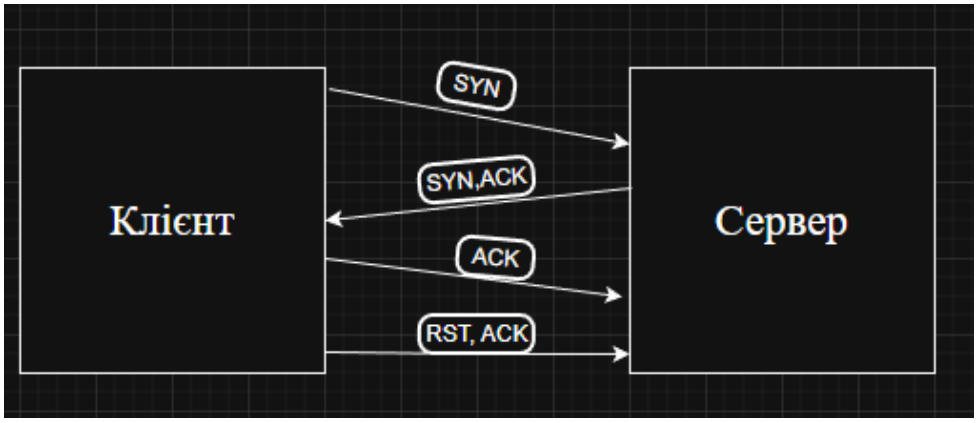


Рисунок 2.2 – Схема сканування TCP-з’єднанням

```
┌──(lapa㉿lapa)-[~/Desktop/diploma-demo]
└─$ sudo nmap localhost -sT
[sudo] password for lapa:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-01 11:09 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000046s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 996 closed tcp ports (conn-refused)
PORT     STATE SERVICE
2222/tcp open  EtherNetIP-1
3000/tcp open  ppp
3306/tcp open  mysql
8080/tcp open  http-proxy

Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds

┌──(lapa㉿lapa)-[~/Desktop/diploma-demo]
└─$
```

Рисунок 2.3 – Результат сканування TCP-з’єднанням

SYN-сканування – це різновид сканування TCP-з’єднанням, яке не потребує завершення тристороннього рукостискання. Замість цього воно розриває з’єднання після того, як відбулось отримання відповіді від сервера. Це корисна особливість, яка зменшує ймовірність того, що сканування буде помічено на цільовій системі і записано в журнал логування, оскільки не було встановлено TCP-з’єднання. Цей тип сканування вибирається, використовуючи команду “nmap -sS target_ip”. Результат виконання такого сканування не буде відрізнятись від результату сканування TCP-з’єднанням, зображеним на рисунку 2.3. Основна різниця між скануванням TCP-з’єднанням та TCP SYN-скануванням полягає в тому, що TCP SYN значно швидший та непомітніший процес але сканування TCP-з’єднанням може бути ефективнішим в ситуаціях, коли у мережі є активний фаєрвол або перевірки на з’єднання. Схема TCP SYN-сканування зображена на рисунку 2.4.

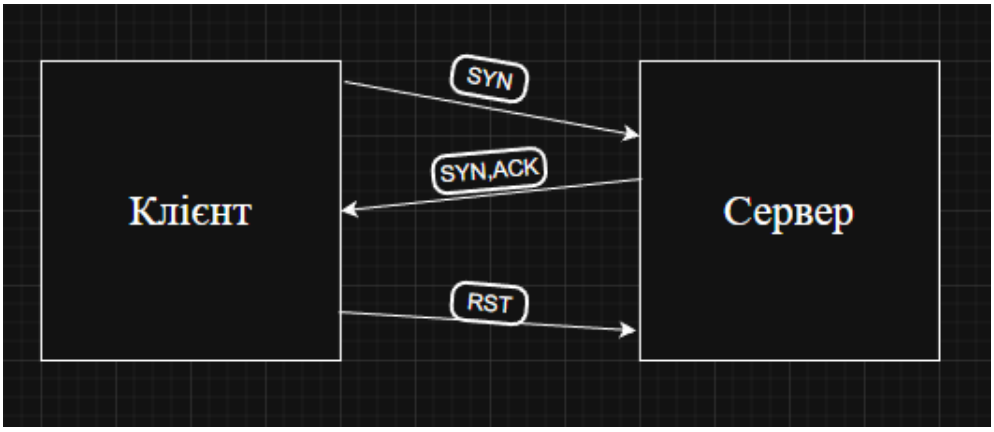


Рисунок 2.4 – Схема TCP SYN-сканування

UDP (User Datagram Protocol) – це протокол, який не потребує жодного рукостискання, адже він працює без встановлення з'єднання. З UDP немає гарантії, що служба, яка прослуховує порт відповість на надіслані пакети.

Під час UDP-сканування Nmap сприймає отримані повідомлення ICMP Destination Unreachable – Port Unreachable Type 3, Code 3 як свідчення про те, що порт закритий. Хоча за відсутності відповіді утворюється стан "open|filtered", оскільки пакети можуть бути відкинуті фаєрволом або такі повідомлення можуть блокуватись чи обмежуватись. Іноді можуть повертатись інші коди, які вказують на мережеву недоступність або фільтрацію трафіку. Отже, при UDP-скануванні нас цікавлять порти UDP, які не генерують жодної відповіді, тобто ті, які Nmap визначить відкритими. Результат виконання UDP-сканування зображено на рисунку 2.5.

```
┌──(lapa㉿lapa)-[~/Desktop/diploma-demo]
└─$ sudo nmap localhost -sU
[sudo] password for lapa:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-01 13:49 UTC
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000020s latency).
Other addresses for localhost (not scanned): ::1
Not shown: 998 closed udp ports (port-unreach)
PORT    STATE         SERVICE
53/udp  open|filtered domain
161/udp open|filtered snmp

Nmap done: 1 IP address (1 host up) scanned in 1.36 seconds

┌──(lapa㉿lapa)-[~/Desktop/diploma-demo]
└─$
```

Рисунок 2.5 – Результат виконання UDP-сканування

Nmap - потужний, незамінний інструмент, але не універсальний. Його використання не дає можливості провести глибокий аналіз логіки інформаційної системи. Результати цього інструменту - перший крок у процесі виявлення, розвідки та сканування, який вимагає подальшої ручної перевірки.

2.3.2 Nessus

Nessus - це один із найбільш відомих інструментів для автоматизованого пошуку вразливостей в інформаційних системах. Сканер здатен аналізувати запущені служби або цілі корпоративні середовища. Інструмент не має відкритого вихідного коду і, здебільшого, надається організацією для більш глибокого аналізу під час тестування на проникнення.[15]

Ідея роботи Nessus полягає у плагінах, які дозволяють перевіряти велику кількість можливих помилок конфігурації, CVE, слабкі налаштування, застарілі сервіси та інші потенційні точки входу в інформаційну систему. Як правило, Nessus використовується у трьох сценаріях. Перший - це регулярна оцінка вразливостей у межах інформаційної інфраструктури, коли сканування запускається за розкладом і дозволяє відстежувати появу нових загроз. Другий - процес виявлення застарілого програмного забезпечення та відсутності оновлення. Третій - допомога у відповідності стандартам безпеки або внутрішнім політикам організації.

В рамках тестування на проникнення Nessus представляє цінність як інструмент, який може сканувати мережу. Є можливість обрати скільки і які порти будуть просканові. Наприклад, сканування всіх портів, або тільки популярні порти, на яких потенційно можуть бути запущені якісь сервіси, або власноруч обрати декілька портів. Інтерфейс Nessus з вибором типів сканування зображено на рисунку 2.6.

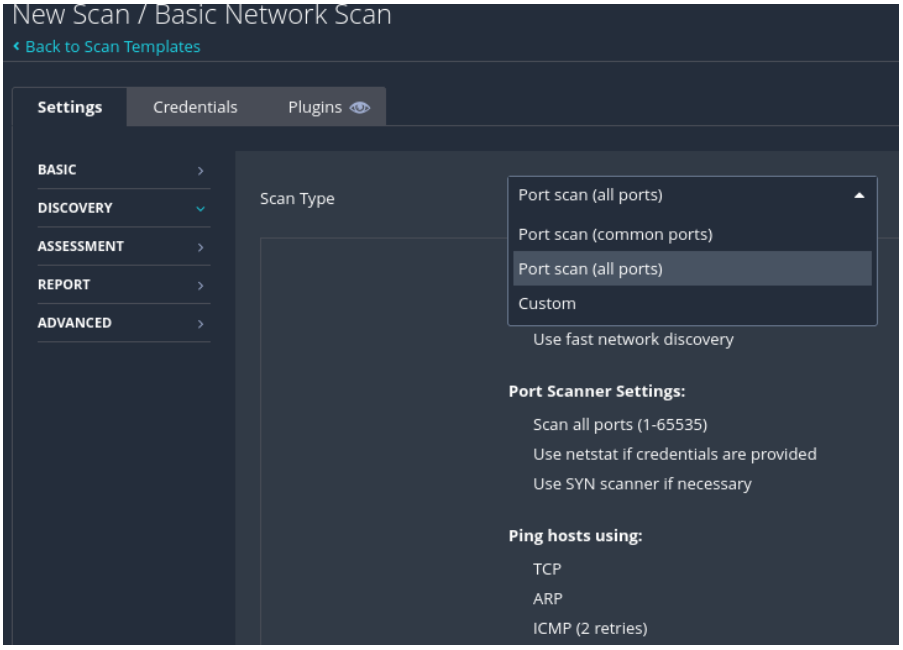


Рисунок 2.6 – Інтерфейс Nessus з вибором типу сканування

Також, Nesssus надає декілька варіантів того, як саме буде відбуватись сканування: сканування на наявність популярних веб-вразливостей, сканування на наявність всіх можливих вразливостей та більш комплексне сканування на наявність всіх вразливостей. На рисунку 2.7 зображено інтерфейс Nessus з вибором як саме має відбуватись сканування.

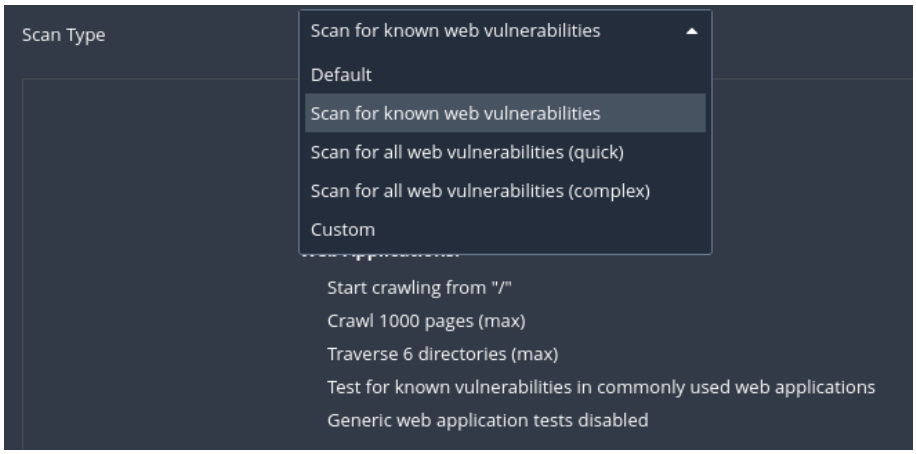
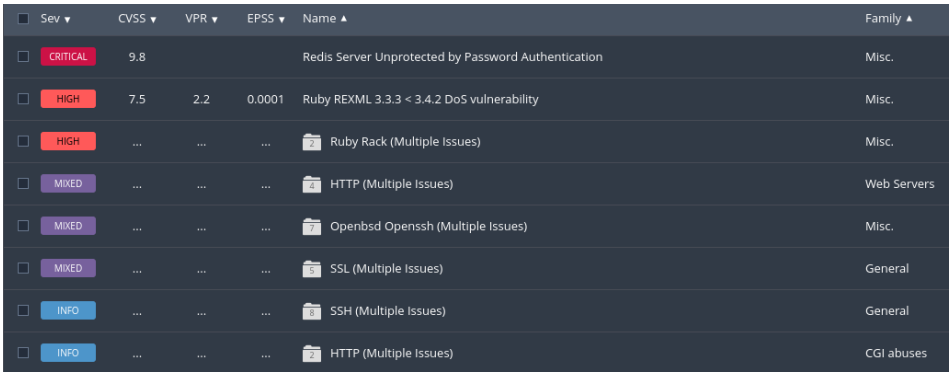


Рисунок 2.7 – Інтерфейс Nessus з вибором процесу типу сканування

Під час сканування, Nessus намагається знайти усі вразливості, які існують на цілі. На графічному інтерфейсі вразливості позначаються за типом критичності різними кольорами: синій - розкриття базової інформації, яка мала б бути конфіденційною, або вдалось визначити версію сервісу; зелений, жовтий та

помаранчевий кольори - це вразливості нижчого, середнього та високого пріоритету різних типів; червоний колір - критичні вразливості, які зазвичай включають в себе віддалене виконання коду або подібні серйозні прогалини в безпеці інформаційної системи. Приклад результату сканування за допомогою Nessus зображений на рисунку 2.8. Круговий графік виявлених вразливостей зображений на рисунку 2.9. Тестування відбувались в локальному середовищі, в якому був запущений сам Nessus, а також додатково були відкриті деякі порти для демонстрації.



Sev	CVSS	VPR	EPSS	Name	Family
CRITICAL	9.8			Redis Server Unprotected by Password Authentication	Misc.
HIGH	7.5	2.2	0.0001	Ruby REXML 3.3.3 < 3.4.2 DoS vulnerability	Misc.
HIGH	...	...	...	Ruby Rack (Multiple Issues)	Misc.
MIXED	...	...	...	HTTP (Multiple Issues)	Web Servers
MIXED	...	...	...	Openbsd Openssh (Multiple Issues)	Misc.
MIXED	...	...	...	SSL (Multiple Issues)	General
INFO	...	...	...	SSH (Multiple Issues)	General
INFO	...	...	...	HTTP (Multiple Issues)	CGI abuses

Рисунок 2.8 – Результат аналізу системи за допомогою Nessus

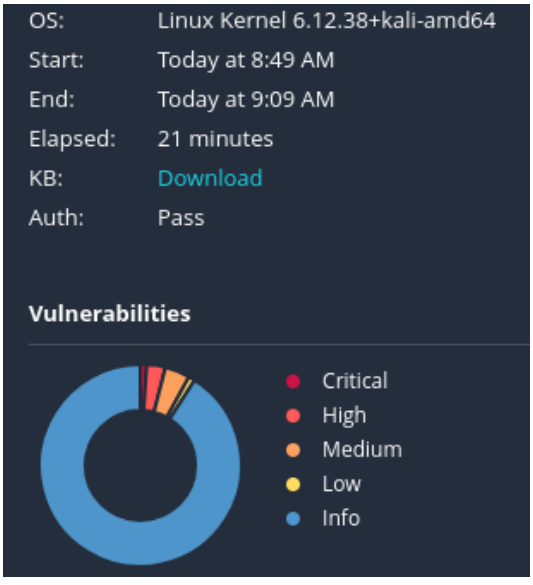


Рисунок 2.9 – Круговий графік знайдених вразливостей за допомогою Nessus

Кожна з виявлених вразливостей має короткий опис, який допомагає визначити подальші кроки тестування та надає базову інформацію про вразливість.

Також, Nessus надає коротку інструкцію щодо усунення вразливості та зазначає на якому порті і на якій IP-адресі її було знайдено. На рисунку 2.10. зображена сторінка опису критичної вразливості, яка була знайдена під час тестування локальної системи.

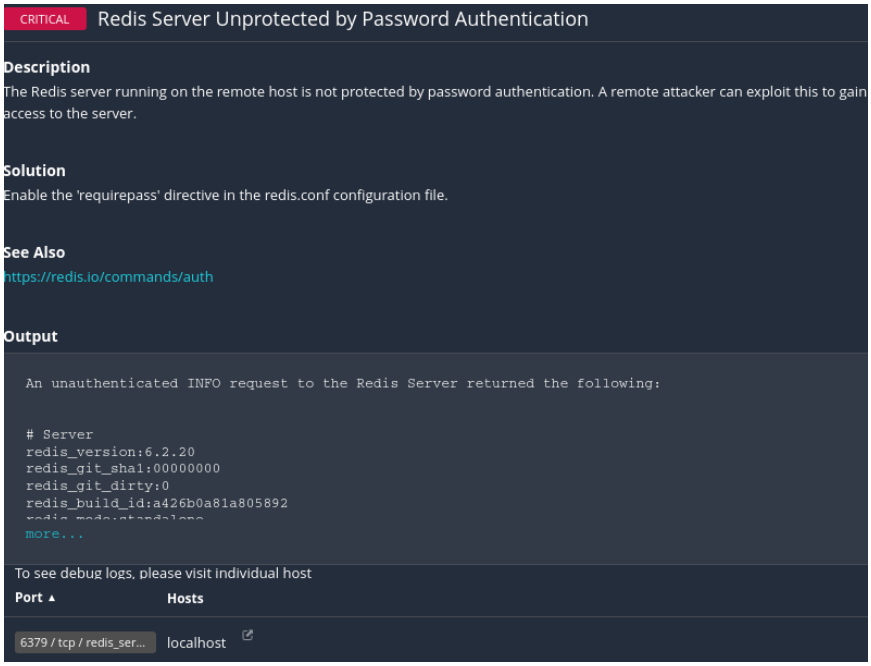


Рисунок 2.10 – Опис виявленої критичної вразливості

Не дивлячись на гнучкість цього інструменту, він не замінює ручне тестування. Це одна з можливих точок входу, інструмент, який виявляє потенційно проблемні місця, які фахівці перевіряють власноруч, комбінуючи сканер з іншими засобами аналізу інформаційних систем та інструментами експлуатації вразливостей або аналізу конфігурації. За своїм функціоналом, Nessus - є невід'ємною частиною проецсу тестування на проникнення через те, що є збірним хабом, який забезпечує широту охоплення, автоматизацію рутинних, щоденних перевірок та об'єктивність у пріоритизації ризиків.

2.3.3 Metsploit Framework

Metasploit Framework - це модульна, консольна платформа для дослідження інформаційних систем та етичного хакінгу. Інструмент дозволяє сканувати систему, збирати інформацію, розробляти та використовувати готові експлойти, реалізовувати процеси для дій після експлуатації. Програма має чітку архітектуру

та можливості програмного розширення, що перетворює одноразові атаки на документовані, відтворюванні процеси, що полегшує перевірку коректності виправлень, аналізу впливу і формування технічних звітів.[14]

Metasploit у процесі аналізу інформаційних систем, зазвичай використовується на етапах сканування, експлаутації, пост-експлуатації та звітності. Такий широкий спектр застосування можливий завдяки тому, що платформа є модульною. Всього модулів сім і кожен модуль використовується за потреби, в залежності від ситуації та системи:

- Auxiliary - допоміжні модулі, які поділені на різні підкатегорії та використовуються для виконання різних допоміжних дій, наприклад сканування, аналіз мереж та систем, підробка запитів, атаки типу відмови обслуговування. Модуль може бути корисним на початкових етапах тестування на проникнення.
- Encoders - модуль, який допомагає в процесі тестування уникнути виявлення антивірусами. Модуль дозволяє закодувати корисне навантаження, зробивши його менш помітним для антивірусного програмного забезпечення.
- Evasion - модуль, який використовується для уникнення виявлення на операційній системі Windows.
- Exploits - програми, які поділені на різні категорії відповідно до програмного забезпечення та містять в собі послідовність команд, кожна з яких направлена на конкретну вразливість.
- Nops - це інструкції для процесору. Як тільки процес зчитує таку інструкцію - він отримує команду “не робити нічого”. Це може бути корисним при переповнені буферу, щоб виділити місце в пам’яті перед тим, як буде виконане корисне навантаження.
- Payloads - модуль з корисним навантаженням, яке було доставлено на атаковану систему. Наприклад, додавання користувача в цільову систему, або запуск якогось додатку, або встановлення мережевого з’єднання між атакуючим та атакованою системою, створення командних оболонок та інше.

- Post - модуль, який використовується після експлуатації цілі. Користь модуля полягає у зборі або викрадені інформації з атакованої системи. Цією інформацією можуть бути: файли, збережені паролі, перелік сервісів та додатків на цілі.

Основний механізм взаємодії із Metasploit - msfconsole. Виконання цієї команди в консолі відкриває інструмент. Результат виконання msfconsole зображено на рисунку 2.11. В консольному режимі інструмент підтримує більшість базових команд операційної системи Linux.

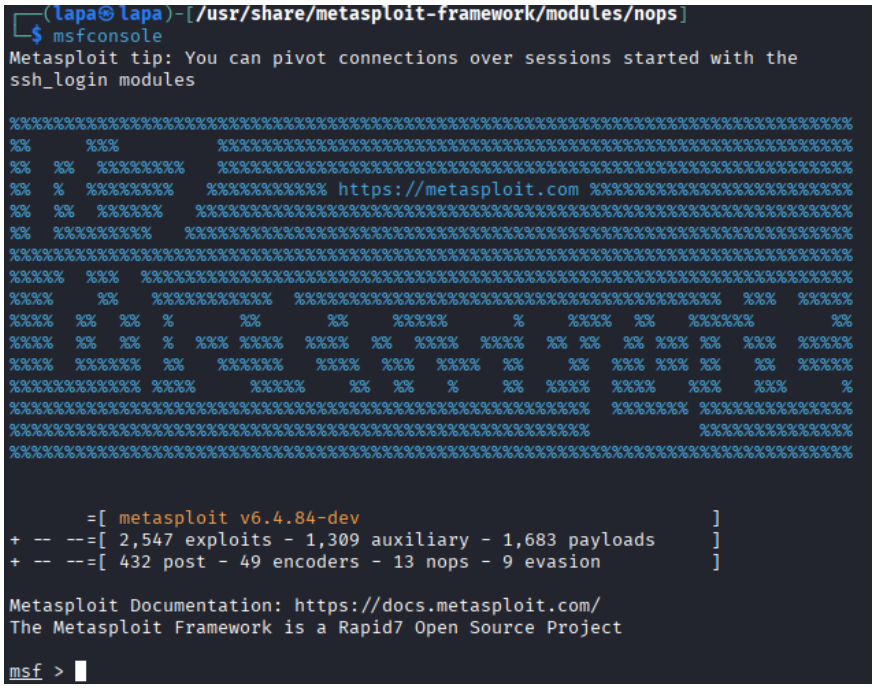


Рисунок 2.11 – Консоль Metasploit Framework – “msfconsole”

Основною метою використання Metasploit - є використання експлойтів та корисного навантаження. Виявивши потенційно вразливий сервіс в інформаційній системі можна використати команду “search". Ця команда шукає в базі даних фреймворку модулі, які відповідають параметру пошуку, наприклад: номер CVE, цільова система або назви експлойтів. На рисунок 2.12 зображено приклад пошуку модулів для ключового слова “ssh/ssh_version”.

```
msf > search ssh/ssh_version

Matching Modules
================

   #  Name
   -  ----
   0  auxiliary/fuzzers/ssh/ssh_version_15
   1  auxiliary/fuzzers/ssh/ssh_version_2
   2  auxiliary/fuzzers/ssh/ssh_version_corrupt
   3  auxiliary/scanner/ssh/ssh_version
```

Рисунок 2.12 – Результати пошуку модулів за параметрами

Процес вибору необхідного модулю залежить від поточної задачі. Обравши модуль, на нього потрібно переключитись. Це можна виконати за допомогою команди “use module_name”. Кожен поточний інструмент в кожному модулі має визначені та невизначені змінні, які можуть бути обов’язковими або не обов’язковими. Для того, щоб перевірити змінні експлойту або інструменту модуля, можна використати команду “info”, знаходячись в каталозі інструменту. В опціях інструменту є: ім’я змінних, поточне налаштування, тобто значення цієї змінної, стовпчик “Required” інформує користувача про те, чи є ця опція необхідною під час роботи інструмента та коротке описання, яке дає підказку, що саме має бути значенням змінної. На рисунку 2.13 зображено вибір сканеру для визначення версії сервісу SSH та перегляд змінних для нього.

```
msf > use auxiliary/scanner/ssh/ssh_version
msf auxiliary(scanner/ssh/ssh_version) > info

       Name: SSH Version Scanner
     Module: auxiliary/scanner/ssh/ssh_version
    License: Metasploit Framework License (BSD)
       Rank: Normal

Provided by:
  Daniel van Eeden <metasploit@myname.nl>
  h00die

Check supported:
  No

Basic options:
  Name             Current Setting  Required  Description
  ----             ---------------  --------  -----------
  EXTENDED_CHECKS  true             yes       Check for cryptographic issues
  RHOSTS                            yes       The target host(s), see https:/
  RPORT            22               yes       The target port
  THREADS          1                yes       The number of concurrent threac
  TIMEOUT          30               yes       Timeout for the SSH probe

Description:
  Detect SSH Version, and the server encryption
```

Рисунок 2.13 – Перелік змінних інструменту сканування SSH

Кожній змінній, в якій значення “Required” вказано - “yes” для коректної роботи інструменту необхідно визначити. Для цього потрібно використати команду “set var_name data”. Наприклад на рисунку 2.13 вказано, що “RHOSTS” має бути визначеним, тож можна використати команду “set RHOSTS localhost” для встановлення віддаленої адреси цілі. На рисунку. 2.14 зображено визначення змінної “RHOSTS” та виконана перевірка на те, що б дані були правильно вставлені.

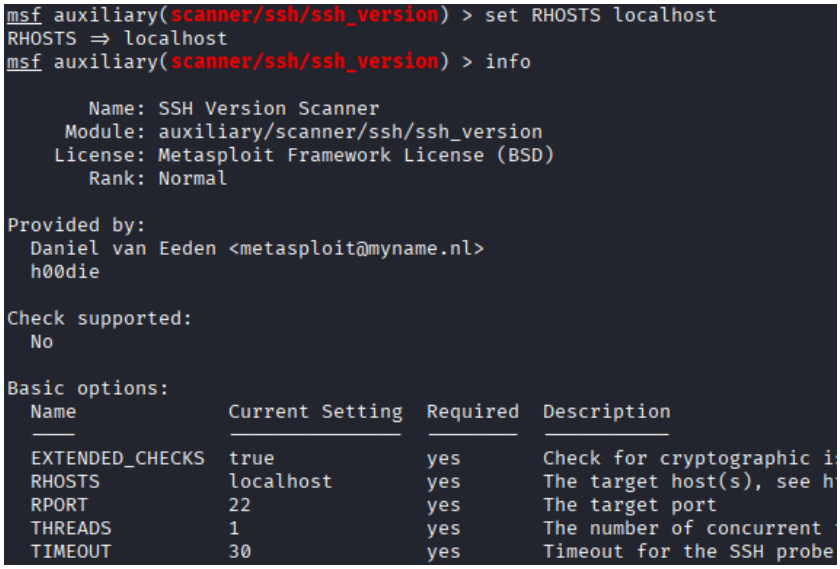


Рисунок 2.14 – Визначення змінної для модулю

Встановивши усі необхідні значення для змінних інструмент потрібно запустити. Це можна зробити за допомогою команди “exploit”. На рисунку 2.15 зображено запуск сканеру версії SSH із допоміжного модулю. Сканування відбувалось локально, на завчасно запущеному, відкритому порті.

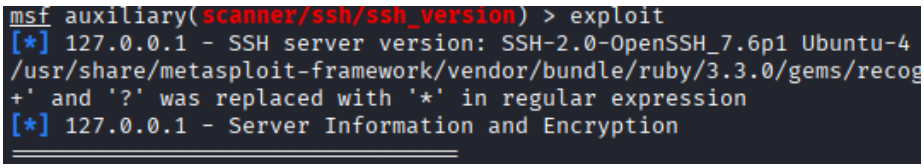


Рисунок 2.15 – Запуск модулю в Metasploit Framework

В більшості випадків, вибір робочого корисного навантаження - це процес спроб і помилок через антивіруси, правила брандмауера, різність середовищ, варіативність розширення файлів та інше. Для підбору в Metasploit Framework

реалізовано роботу з сесіями, тобто - можливість переводити сеанс у фоновий режим. Це може бути корисним при роботі з однією кінцевою точкою або під час роботи над кількома цілями одночасно.

Metasploit Framework – це ефективний модульний інструмент для практичної верифікації вразливостей. Платформа цінна тим, що має стандартизовані експлуатаційні сценарії, прискорює підтвердження ризиків та дозволяє відтворити тести під час аналізу вразливостей інформаційної системи.

Загалом, інструменти для етичного тестування інформаційних систем на проникнення – це невичерпний список програм, скриптів та додатків, які використовуються в залежності від ситуації, системи, правил тестування, сканування та багатьох інших факторів. Деякі інструменти є неспеціалізованими для етичного хакінгу, а скоріше адаптовані під необхідні задачі, деякі не можуть вирішити нагальних потреб і вимагають доробки, написання модулів, реалізацію додаткових сервісів. Якщо інструментарій не дозволяє розширення, процес етичного хакінгу доповнюється розробкою уніфікованих, спеціалізованих інструментів під конкретну задачу. Розглянуті інструменти – це базовий мінімум, визначений спільнотою, необхідний для ручного тестування на проникнення, який дозволяє імітувати дії злочинців та забезпечувати якісний аналіз вразливостей інформаційних систем.

2.4 Автоматизація та інтеграція штучного інтелекту

В умовах сьогоднішніх тенденцій, штучний інтелект стає невід’ємною частиною більшості процесів, особливо в сфері інформаційних технологій. Це стосується і процесів, пов’язаних з тестуванням інформаційних систем на проникнення. Основний вплив такої автоматизації на пентестинг проявляється у реалізації додаткових методів сканування.

Pantera, Cymulate або PenTest++ - інструменти, які можуть сканувати порти або сервіси, перевіряти конфігурації та збирати усю поверхню атаки. Але, не дивлячись на те, що автоматизовані сканери дозволяють швидше отримати розуміння того, що відбувається у системі, необхідність у ручній валідації

отриманих даних нікуди не зникає. Проблема полягає в тому, що на сьогоднішній день штучний інтелект не досконалий. В контексті тестування на проникнення виникає дві серйозні проблеми:

- Питання конфіденційності отриманих даних
- Проблема “галюцинацій” штучного інтелекту

Скануючи або тестуючи інформаційну систему, існує ризик того, що дані можуть зберігатись у хмарному середовищі автоматизованого інструменту. В більшості випадків, організації не займаються створенням власних продуктів, а купують ліцензійне програмне забезпечення у сторонніх виробників. Тим самим, під час використання такого програмного забезпечення, перекладають відповідальність за безпеку отриманих даних на розробників. Ризик полягає в тому, що в процесі сканування не існує гарантії, що в разі витоку даних з автоматизованого інструменту, чутливі дані залишаться конфіденційними. Тобто, організація, яка довірила чутливі дані сторонньому інструменту і не була ціллю атаки може постраждати.

Проблема “галюцинацій” штучного інтелекту поширена в будь-якому контексті, в якому згадується сам штучний інтелект. Не існує гарантій, що інструмент автоматизації, проаналізувавши дані буде надавати чіткі, правильні, безпечні поради. З цієї ж причини, не можна довіряти роботу таких інструментів в середині інфраструктури, дозволяючи виконувати будь-які дії. Враховуючи це, інструмент автоматизації має бути допоміжним, а не основним програмним забезпеченням. Також, за таких умов, ярко підкреслюється роль експертного підходу в питанні застосування інструментів автоматизації за допомогою штучного інтелекту.

В межах наукового дослідження було розроблено прототип модульної інтеграції результатів сканування та аналізу штучного інтелекту мовою програмування Python. На відміну від звичайних поширених сканерів, інструмент дає можливість скоротити витрачений час на виявлення потенційних вразливостей, надаючи конкретні поради щодо знайдених сервісів, портів та програмного забезпечення загалом. Інструмент використовує результати сканування Nmap,

відправляє їх на API штучного інтелекту, розробленого Google – Gemini та повертає проаналізовані дані з наступними потенційними кроками.

До ключових модулів реалізації належать:

- Модуль конфігурації та формування системного запиту до моделі (prompt);
- Модуль нормалізації даних у формат XML для уніфікації даних;
- Модуль інтеграції з API Gemini, який, використовуючи спеціальний ключ, підключається до хмарного сервісу штучного інтелекту;
- Модуль обробки відповіді від моделі, який виокремлює дані та приводить їх в уніфікований вид.
- Модуль генерації звіту, який записує отримані уніфіковані дані у файл.

На рисунку 2.16 зображена архітектура інструменту.

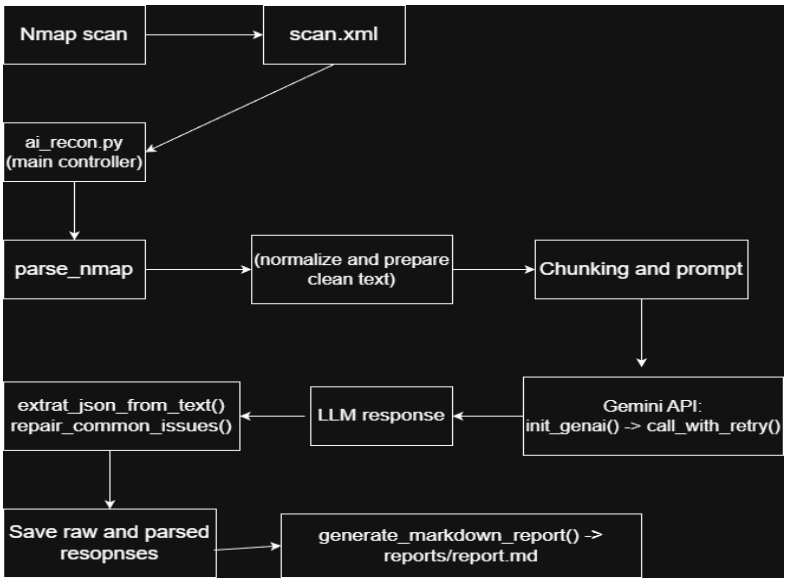


Рисунок 2.16 – Архітектура інструменту із інтеграцією штучного інтелекту

Програмний код реалізації такої інтеграції наведений у додатку А. Приклади результатів роботи інструменту зображені на рисунку 2.17 та на рисунку 2.18. Тестування проводилось у локальній мережі для безпеки та етичності процесу дослідження.

```
  "summary": "The scan reveals a system with numerous outdated and potentially vulnerable services, including FTP, SSH, Telnet, Samba, and web servers. Several services are running with known vulnerabilities that could lead to unauthorized access, privilege escalation, or denial of service.",
  "findings": [
    {
      "title": "Outdated vsftpd FTP Server",
      "severity": "Critical",
      "evidence": "192.168.0.104 21/tcp open ftp vsftpd 2.3.4 None",
      "steps_to_verify": [
        "Attempt anonymous login to the FTP server.",
        "Check for known vulnerabilities in vsftpd 2.3.4, such as CVE-2011-2523 (command execution)."
      ],
      "remediation": "Upgrade vsftpd to the latest stable version. If an upgrade is not immediately possible, disable anonymous access and ensure strong authentication mechanisms are in place. Restrict access to the FTP service to only necessary IP addresses."
    },
```


Рисунок 2.18 – Результат аналізу сканування за допомогою ШІ збережений у форматі “.json”


```
This network scan reveals a host, 192.168.0.104, with a significantly outdated and insecure configuration, presenting a high-risk profile. The sheer number of open services, many of which are legacy or known to have vulnerabilities, indicates a severe lack of patching and security hardening. The presence of services like `vsftpd 2.3.4`, `OpenSSH 4.7p1`, `Linux telnetd`, `Samba smbd 3.X - 4.X`, `netkit-rsh rexecd`, `bindshell Metasploitable`, and `UnrealIRCd` are immediate red flags. These versions are known to be vulnerable to a wide array of exploits, including remote code execution, information disclosure, and denial-of-service attacks.

The outdated FTP server (`vsftpd 2.3.4`) is particularly concerning. This version is notoriously vulnerable to anonymous login exploits and has known security flaws that can be leveraged for unauthorized access and file manipulation. Similarly, the `OpenSSH 4.7p1` version is old and may be susceptible to brute-force attacks or known vulnerabilities that could allow for credential compromise. The presence of `telnetd` is another critical issue, as it transmits credentials in plain text, making it trivial for an attacker to intercept sensitive information.

The Samba services (`smbd 3.X - 4.X` and `3.0.20-Debian`) also pose a significant risk. Older versions of Samba are prone to vulnerabilities that can lead to remote code execution and unauthorized access to file shares. The `netkit-rsh rexecd` service, along with `login` and `tcpwrapped` on ports 512, 513, and 514 respectively, are remnants of older, insecure remote administration protocols that should be disabled. The `bindshell Metasploitable` on port 1524 is a direct indicator of a potentially compromised or intentionally vulnerable system, likely part of a penetration testing lab, but if found in a production environment, it's a critical security breach.
```


Рисунок 2.18 – Результат аналізу сканування за допомогою ШІ збережений у форматі “.txt”

Інструмент подібного характеру може бути корисним для того, щоб автоматизувати та пришвидшити процес сканування інформаційної системи на вразливості. Метою розробки було протестувати можливість практичної реалізації інструменту, який здатен фоновим процесом аналізувати та радити наступні кроки під час тестування.

Але подібна реалізація містить вагомі недоліки. Інструмент залежить від ключа API, який зберігається локально на хості. В разі компрометації серверу, який містить в собі ключ, зловмисник буде здатен отримати майже готовий процес злому системи. Також, більшість моделей генеративного штучного інтелекту отримують та надсилають дані токенами. Токен, в контексті штучного інтелекту – це розмір даних, який задається вручну. Чим більше значення мають токени – тим більше

навантаження йде на модель і тим більше це може споживати ресурсів, як фактичних, так і апаратних. Через це немає гарантії що аналіз буде успішним кожен раз, коли він необхідний. Основна проблема, яка була виявлена під час розробки, – це відсутність можливості отримати більше одного токену від моделі, тож аналіз потрібно розбивати на менші частини. Також, проблема безпеки даних не вирішена, адже інформація про систему надсилається на сторонній сервіс, що може стати потенційною вразливістю.

Однак інструмент подібного функціоналу має широкий потенціал розвитку. Завжди є можливість додати інші типи сканерів, щоб автоматизувати обробку різних даних. Функціонал такого додатку може бути використаним і поза етичним тестуванням. Наприклад, можна додати інтеграцію з SIEM-системами. Також, налаштувавши правильну обробку вхідних та вихідних даних, можна винести скрипт із консолі та розробити окремий додаток. І найголовніше – розробка спеціалізованої моделі, яка буде навчена для виконання завдань в рамках кібербезпеки та етичного тестування. Це дозволить підвищити точність отриманих рекомендацій та захистити дані. Через те, що така модель може бути розгорнута локально, в рамках організації, на хості, який буде поза мережею проблема безпеки сканування може бути вирішеною.

ВИСНОВОК ДО РОЗДІЛУ 2

В даному розділі була в деталях проаналізована практична частина процесу етичного тестування на проникнення інформаційних систем. Були чітко виділені етапи тестування, які формують логічний ланцюг дій, що гарантує ефективність процесу. Завчасне узгодження меж тесту допомагає виявити технічні недоліки під час пентестингу. Сканування та аналіз виявляють потенційні точки входу, або ж формують уявлення про те, як до них дібратись. Експлуатація та пост-експлуатація дозволяють ефективно оцінювати вплив вразливостей на конфіденційність, цілісність та доступність інформаційної системи. Фінальний етап - звітування допомагає перетворити технічні знахідки у структурований список проблемних місць, або ж надати практичні рекомендації для управління ризиками.

Були розглянуті методології тестування, які виступають в якості каркасу, в якому сформовано єдиний набір очікувань для виконавців та замовників. Важливість полягає в тому, що це не стільки вимога конкретних дій, скільки правила взаємодії, в яких вказано загальним образом, що дозволено, які поточні критерії успіху та як саме результати будуть верифіковані. Окрема перевага полягає у можливості комбінувати підходи. Наприклад, існує можливість взяти за основу PTES для організаційної складової, додати OSSTMM для охоплення більш різноманітних аспектів та, за необхідності, використати OWASP для аналізу веб-компонентів.

Інструменти на кшталт Nmap, Nessus та Metasploit Framework перетворюють методологічні вимоги на практичну реалізацію. Nmap дає первинну картину мережі, Nessus дає виявляти відомі вразливості, а Metasploit дозволяє перетворювати теоретичні знахідки на відтворювані сценарії експлуатації.

Автоматизація може суттєво змінити підхід до тестування на проникнення. Впровадження автоматизованих методів сканування та інтеграція з системами управління вразливостями підвищує швидкість виявлення та аналізу. Однак автоматизація не відміняє необхідність етичності процесу. Наявність LLM-моделей відкриває нові можливості для масштабування досліджень, але це супроводжується ризиками неправильних висновків і потенційного зловживання. В цьому контексті,

найбільш ефективною стратегією є поєднання автоматизації за допомогою генеративного штучного інтелекту та автоматичними інструментами, які дозволяють проводити процес сканування під наглядом і швидко інтерпретувати вихідні дані.

Тож, ефективне тестування на проникнення є не просто набором інструментів або переліком етапів, а комплексним підходом, який поєднує методологічну дисципліну, технічні можливості та помірковану автоматизацію. Успішність практики вимірюється не тільки знайденим вразливостями, а їхньою верифікацією, рекомендаціями щодо подальших дій та здатністю адаптувати власні дії під конкретну задачу.

3 ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ МЕТОДАМИ ЕТИЧНОГО ХАКІНГУ

3.1 Опис використаних систем та процес їх встановлення

3.1.1 Kali Linux

Найпоширенішою операційною системою для пентестингу є Kali Linux. Це спеціалізований дистрибутив, який створений під завдання інформаційної безпеки. У нього включено широкий спектр встановлених інструментів для всіх етапів тестування. Ідея використання даного дистрибутиву полягає в тому, щоб створити контрольоване середовище, в якому інструменти доступні одразу після встановлення системи, а тестувальник може швидко переходити між різними етапами етичного хакінгу. Як і будь-яка операційна система, Kali можна розгорнути в окремому віртуальному середовищі. Це дозволяє, в разі необхідності відновити стан лабораторії після тестів і мінімізувати ризики, які можуть вплинути на основну систему.[13]

В якості віртуального середовища, зручно використовувати Oracle Virtual Box. Доступ до безпечної версії цього програмного забезпечення відбувається за посиланням [https://www.virtualbox.org/]. Безпечний доступ до дистрибутиву Kali Linux відбувається за посиланням [https://www.kali.org/].

Після завантаження та встановлення усього програмного забезпечення потрібно створити віртуальну машину. Для цього має бути встановлений ISO-образ Kali Linux. У Virtual Box, у вкладці створення нової віртуальної машини необхідно визначити ім’я операційної системи, де буде знаходитись основний диск, який образ операційної системи буде використаний для встановлення та дистрибутив операційної системи. На рисунку 3.1 зображений інтерфейс Virtual Box із необхідними встановленими опціями.

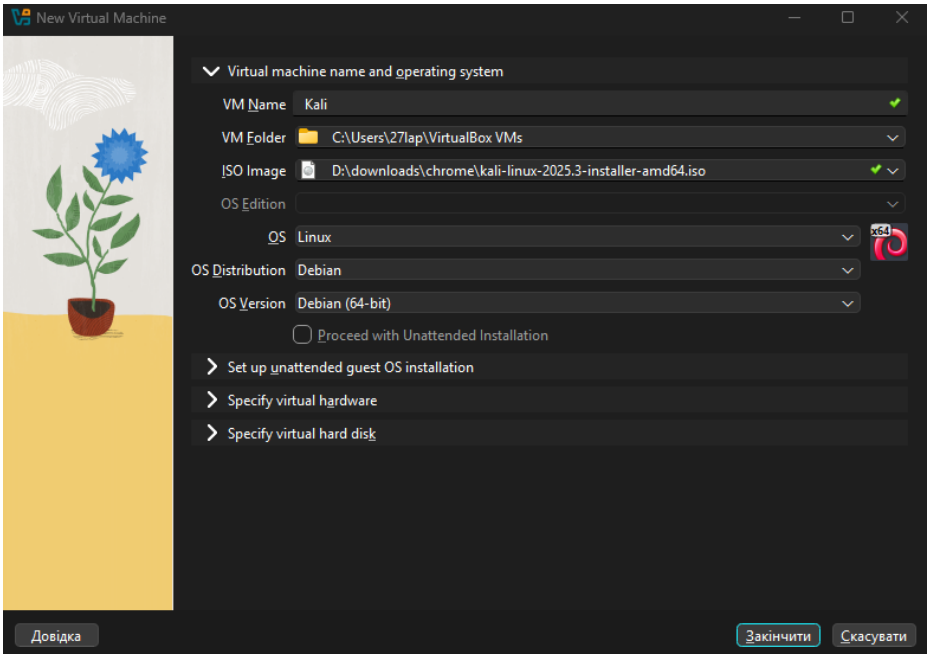


Рисунок 3.1 – Визначення імені, образу та дистрибутиву

Наступним кроком буде визначення кількості оперативної пам’яті та кількість встановлених ядер в системі. Це дозволить системі залишатись достатньо швидкою при роботі з декількома програмами одночасно. На рисунку 3.2 зображено процес встановлення пам’яті та ядер процесору.

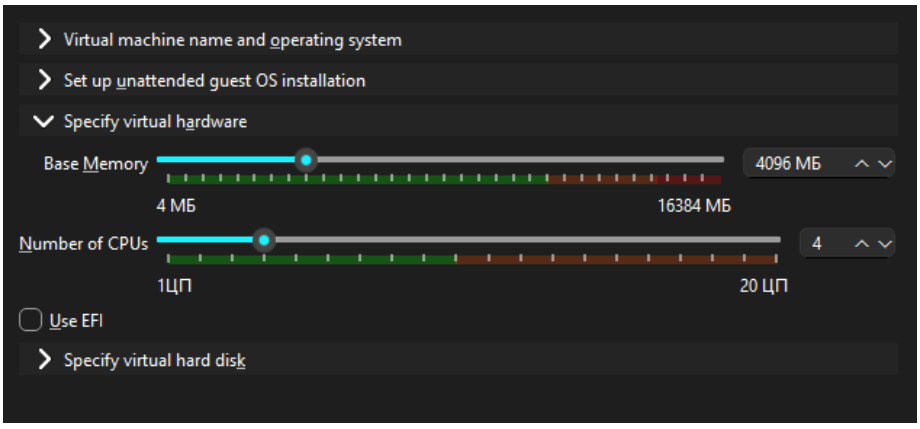


Рисунок 3.2 – Встановлення оперативної пам’яті та кількості ядер

Далі необхідно визначити кількість фактичної пам’яті на основній операційній системі, яка буде виділена під пам’ять віртуальної машини. На рисунку 3.3 зображено інтерфейс Virtual Box із вибором кількості пам’яті для Kali Linux. Після виконання цих дій, відбувається стандартне встановлення операційної системи і Kali Linux готова до використання.

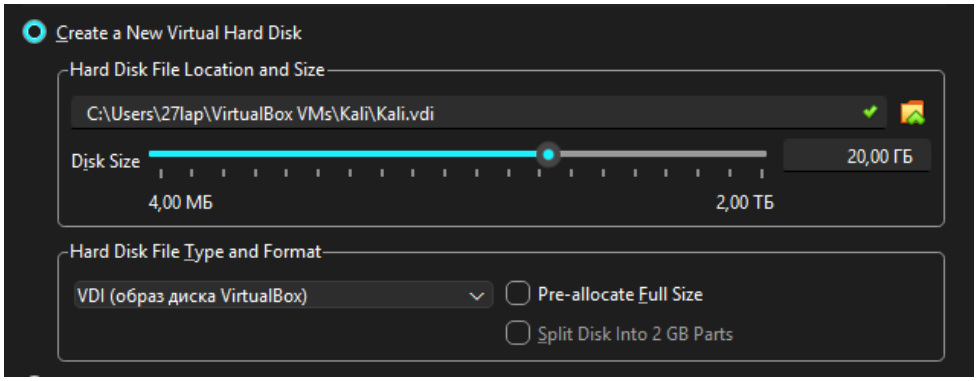


Рисунок 3.3 – Вибір кількості фактичної пам’яті для Kali Linux

3.1.2 Metasploitable

Для того, щоб залишатися в рамках етичності процесу пентестингу, була використана тренувальна вразлива віртуальна машина Metasploitable. На відміну від Kali, Metasploitable - це ціль. Тобто, набір сервісів із спеціально вмонтованими вразливостями або з вразливими сервісами, який дозволяє відпрацювати сценарії атак без порушення законів. Так само, як і Kali Linux, Metasploitable встановлена як віртуальна машина, імітуюча діяльність серверу. Доступ до образу пам’яті виконується за посиланням: [https://www.rapid7.com/products/metasploit/metasploitable/].

При розгортанні лабораторії дії були подібними до тих, які були виконані при розгортанні Kali Linux. Були встановлені ім’я та місце, де буде знаходитись директорія з пам’яттю машини. Але, на відміну від Kali Linux, для встановлення цієї віртуальної машини було використано образ пам’яті, а не ISO-образ. Це означає, що всі налаштування, програми та сервіси були встановлені автоматично, тож замість вказування ISO-образу, в додаткових налаштуваннях Virtual Box було встановлено образ пам’яті. На рисунку 3.4 зображено інтерфейс визначення імені та шлях до директорії пам’яті.

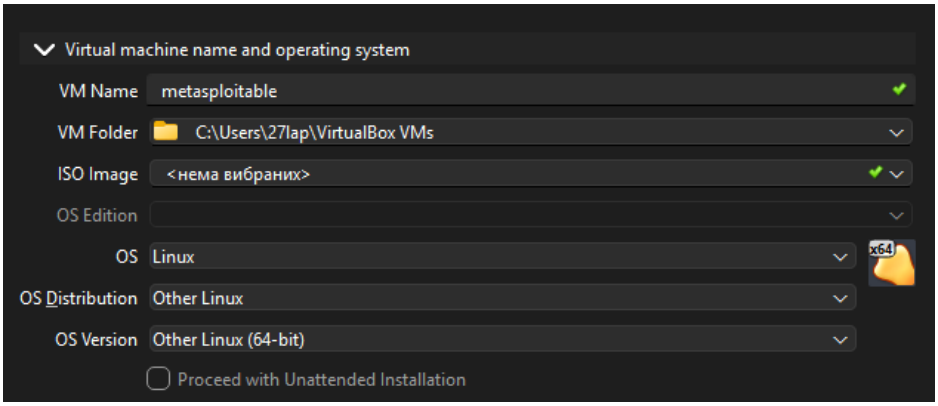


Рисунок 3.4 – Визначення імені, шляху для зберігання диску та операційної системи

Наступним кроком було встановлено кількість оперативної пам’яті та кількість виділених ядер процесору з фізичної машини. Враховуючи, що Metasploitable не буде виконувати жодних дій, кількість була визначена значно менша. На рисунку 3.5 зображено визначення оперативної пам’яті та кількості ядер.

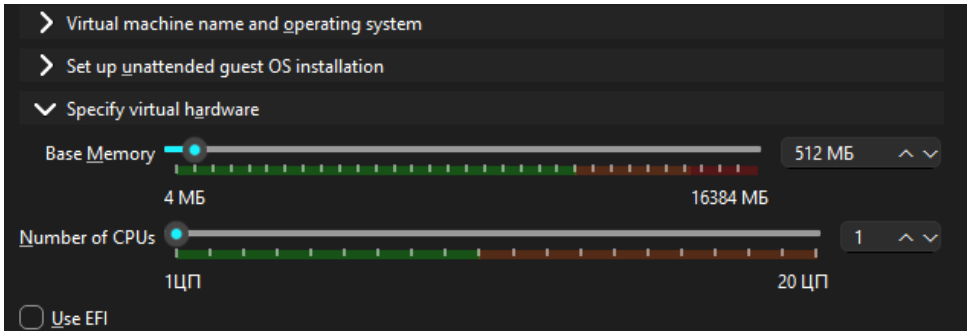


Рисунок 3.5 – Встановлення оперативної пам’яті та кількості ядер

Під час визначення фактичної пам’яті була обрана опція використання існуючого диску. Обраний диск - це диск, який був отриманий з офіційного сайту. На рисунку 3.6 зображено інтерфейс Virtual Box із вибором існуючого, завантаженого диску.

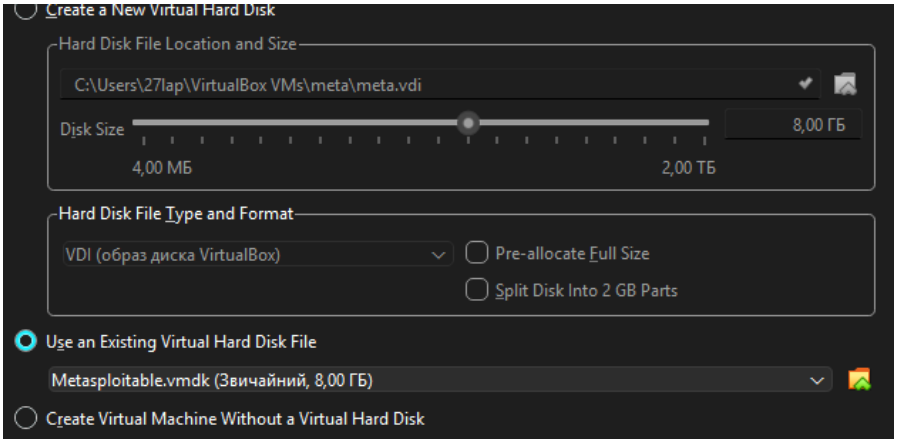


Рисунок 3.6 – Встановлення підготовленого віртуального диску

Важливим елементом, який не можна пропускати є те, що необхідно встановити тип мережі для Metasploitable на “проміжний адаптер” (bridged adapter). Це потрібно для того, щоб машина належала до локальної мережі.

Після завершення встановлення та автоматичного налаштування доступ до віртуальної машини встановлюється за загальнодоступними обліковими даними, де логін: “msfadmin” та пароль “msfadmin”. Після введення облікових даних можна потрапити в систему під користувачем “msfadmin”. Для перевірки цього, на рисунку 3.7 зображено виконання команди “whoami”, яка демонструє ім’я поточного користувача.

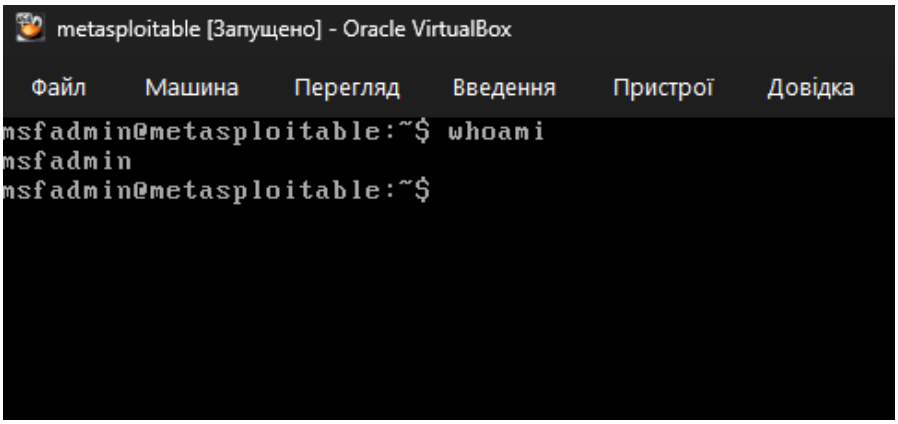


Рисунок 3.7 – Демонстрація успішної автентифікації

Для подальших тестів в директорії користувача було створено файл з назвою “flag.txt”. Під час тестування метою буде прочитати вміст файлу. На рисунку 3.8 зображено вміст файлу.

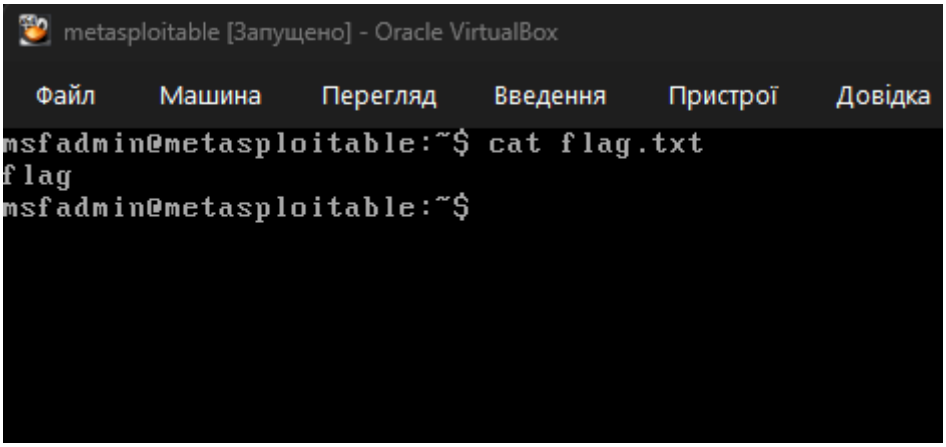


Рисунок 3.8 – Зміст підготовленого файлу flag.txt

Встановлення цієї віртуальної машини зробить практичну демонстрацію етичного тестування та аналізу системи безпечною та етичною.

3.2 Проведення аналізу вразливостей інформаційної системи Metasploitable

3.2.1 Розвідка та визначення потенційних вразливостей

Перед початком тестування була визначена локальна IP-адреса цілі. На рисунку 3.9 зображено виконання команди “ifconfig”, яка виводить дані мережі та IP-адресу.

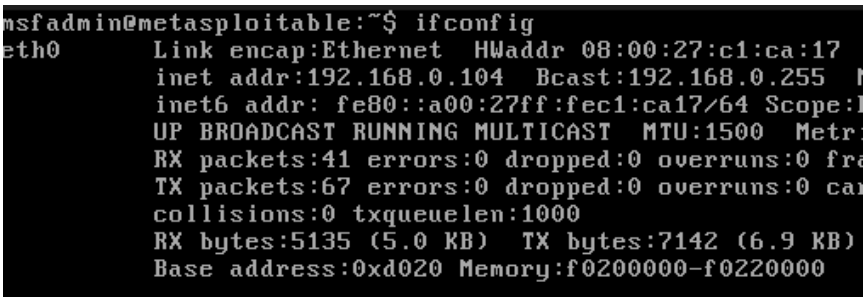


Рисунок 3.9 – Визначення IP-адреси цілі

Первинне сканування відбувається за допомогою утиліти Nmap. Було виконано сканування TCP-з’єднанням із встановленим прапором “-sT”. Результатом має бути вивід команди з демонстрацією відкритих портів на хості. На рисунку 3.10 зображено сканування TCP-з’єднанням та його результати.

```
└─$ sudo nmap 192.168.0.104 -sT
[sudo] password for lapa:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-09 11
Nmap scan report for 192.168.0.104
Host is up (0.0050s latency).
Not shown: 977 filtered tcp ports (no-response)
PORT     STATE SERVICE
21/tcp   open  ftp
22/tcp   open  ssh
23/tcp   open  telnet
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
512/tcp  open  exec
513/tcp  open  login
514/tcp  open  shell
1099/tcp open  rmiregistry
1524/tcp open  ingreslock
2049/tcp open  nfs
2121/tcp open  ccproxy-ftp
3306/tcp open  mysql
5432/tcp open  postgresql
```

Рисунок 3.10 – Сканування TCP-з’єднанням

Наступним кроком сканування є використання інструменту Nessus. На рисунку 3.11 зображено результат сканування Nessus.

Sev	CVSS	VPR	EPSS	Name
CRITICAL	10.0 *	7.4	0.87	UnrealIRCd Backdoor Detection
CRITICAL	10.0			Canonical Ubuntu Linux SEoL (8.04.x)
CRITICAL	10.0 *			VNC Server 'password' Password
CRITICAL	9.8			SSL Version 2 and 3 Protocol Detection
CRITICAL	9.8			Bind Shell Backdoor Detection
MIXED	...	...	...	Apache Tomcat (Multiple Issues)
CRITICAL	...	...	...	SSL (Multiple Issues)
HIGH	7.5	5.9	0.7993	Samba Badlock Vulnerability
HIGH	7.5			NFS Shares World Readable

Рисунок 3.11 – Результат сканування за допомогою Nessus

Додатковим кроком було використання сканеру, який аналізує вивід Nmap та дає поради на основі штучного інтелекту. На рисунку 3.12 зображено приклад аналізу виводу штучним інтелектом.

```
    {
      "title": "Outdated OpenSSH 4.7p1",
      "severity": "High",
      "evidence": "192.168.0.104 22/tcp open ssh OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0",
      "steps_to_verify": [
        "Attempt to connect to the SSH service on port 22.",
        "Perform version detection to confirm the OpenSSH version.",
        "Research known vulnerabilities for OpenSSH 4.7p1, such as potential brute-force attacks or
privilege escalation vectors if weak credentials are used."
      ],
      "remediation": "Upgrade OpenSSH to the latest stable version. Older versions may contain
vulnerabilities that allow for unauthorized access, denial of service, or privilege escalation.
Implement strong password policies, consider disabling password authentication in favor of SSH keys, and
configure fail2ban or similar intrusion prevention systems."
    },
```


Рисунок 3.12 – Приклад аналізу за допомогою штучного інтелекту

Виконавши поверхневий та глибокі аналізи, була отримана інформація про відкриті порти, вразливі сервіси та недоліки, які роблять ціль вразливою. Також, за допомогою штучного інтелекту та інструменту Nessus були отримані додаткові дані про ціль.

3.2.2 Процеси експлуатації та пост-експлуатації

З отриманих даних найцікавішими є критичні вразливості з Nessus, які зображені на рисунку 3.11. Першою потенційною ціллю є VNC Server, в якому встановлений пароль “password”. Якщо результат буде успішним - буде отримано віконну сесію на цільовій машині. Для початку потрібно переконатись, що підключення вразливе. Для цього було використано сканер з Metasploit Framework – “vnc_login”. На рисунку 3.13 зображено вибраний сканер із встановленими змінними “PASSWORD”, “RHOSTS”, “RPORT”.

```
PASSWORD          password                          no
PASS_FILE         /usr/share/metasploit-framewo     no
                  rk/data/wordlists/vnc_passwor
                  ds.txt
Proxies                                             no


RHOSTS            192.168.0.104                     yes


RPORT             5900                              yes
```

Рисунок 3.13 – Визначення змінних у модулі сканеру

На рисунку 3.14 зображено виконання додаткового сканеру “vnc_login” для визначення чи є підключення.

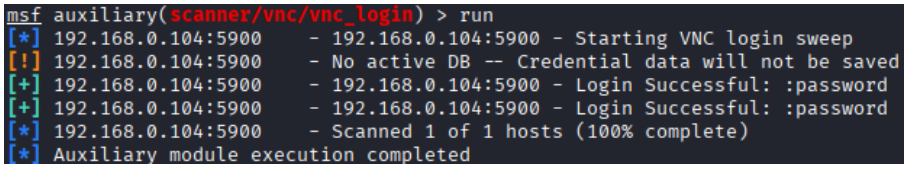


Рисунок 3.14 – Виконання додаткового сканеру vnc_login

Враховуючи, що сканер із встановленим паролем “password” успішно підключається до системи, можна виконати команду “vncviewer 192.168.0.104:5900”, яка має відкрити сесійне вікно. На рисунку 3.15 зображено виконання команди “vncviewer ip:port”.

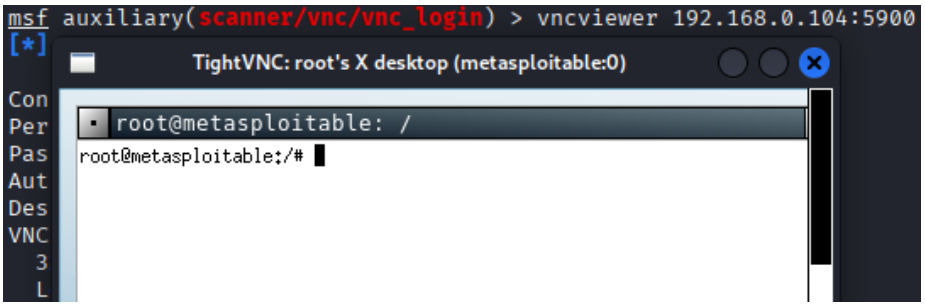


Рисунок 3.15 – доступ до цільової системи

Результат виявився навіть краще ніж очікувалось, адже підключення відбулось до користувача “root”. Це означає, що було отримано найвищі привілеї в системі. На рисунку 3.16 зображено процес пост-експлуатації під час якого було отримано доступ до завчасно створеного файлу “flag.txt”.

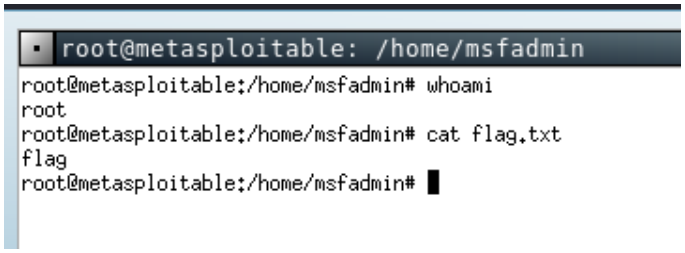


Рисунок 3.16 – Процес пост-експлуатації

Наступну вразливість було проаналізовано за порадою автоматизованого інструменту з інтеграцією штучного інтелекту - VSFTPD сервіс. На хості було виявлено вразливу версію ftp-сервісу. Мінімальний позитивний результат тестування - доступ до системи. Для початку в Metasploit Framework було знайдено

доступні експлойти цієї версії та визначено змінні “RHOSTS” та “RPORT”. На рисунку 3.17 зображено модуль експлуатації та визначені опції для версії 2.3.4 сервісу “vsftpd”.

```
msf exploit(unix/ftp/vsftpd_234_backdoor) > options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   CHOST                     no        The local client address
   CPORT                     no        The local client port
   Proxies                   no        A proxy chain of format typ
                                       proxies: sapni, socks4, soc
   RHOSTS   192.168.0.102    yes       The target host(s), see htt
                                       t/basics/using-metasploit.h
   RPORT    21               yes       The target port (TCP)
```

Рисунок 3.17 – Експлойт для вразливої версії “vsftpd”

На рисунку 3.18 зображено результат виконання експлойту для вразливої версії ftp-сервісу. Було виконано команду “whoami” та знайдений флаг для демонстрації пост-експлуатації.

```
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
[*] 192.168.0.102:21 - Banner: 220 (vsFTPd 2.3.4)
[*] 192.168.0.102:21 - USER: 331 Please specify the password.
[+] 192.168.0.102:21 - Backdoor service has been spawned, handling...
[+] 192.168.0.102:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 2 opened (10.0.2.15:39135 → 192.168.0.102:6

whoami
root
pwd
/
cd home/msfadmin
cat flag.txt
flag
```

Рисунок 3.18 – Експлуатація та пост-експлуатація ftp-сервісу

Наступна вразливість знаходиться на поверхні. Для отримання доступу до системи було використано сервіс telnet. У системах, в яких є відкриті порти telnet є можливість спробувати отримати доступ за допомогою команди “telnet IP”. На рисунку 3.19 зображена спроба отримати доступ і сама вразливість. В консолі під час спроби входу, виводяться облікові дані для отримання доступу до системи.

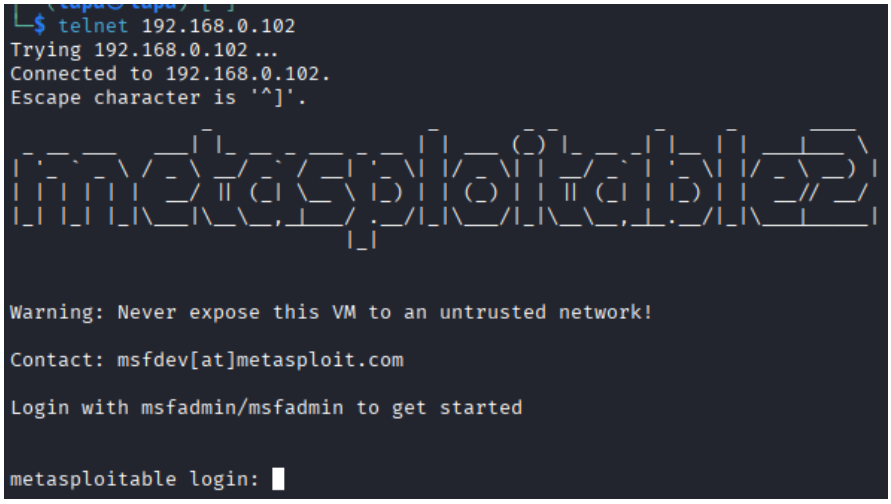


Рисунок 3.19 – Спроба отримати доступ через telnet

На рисунку 3.20 зображено процес пост-експлуатації. Було отримано доступ до файлу “flag.txt” та виконано команду для виявлення прав поточного користувача.

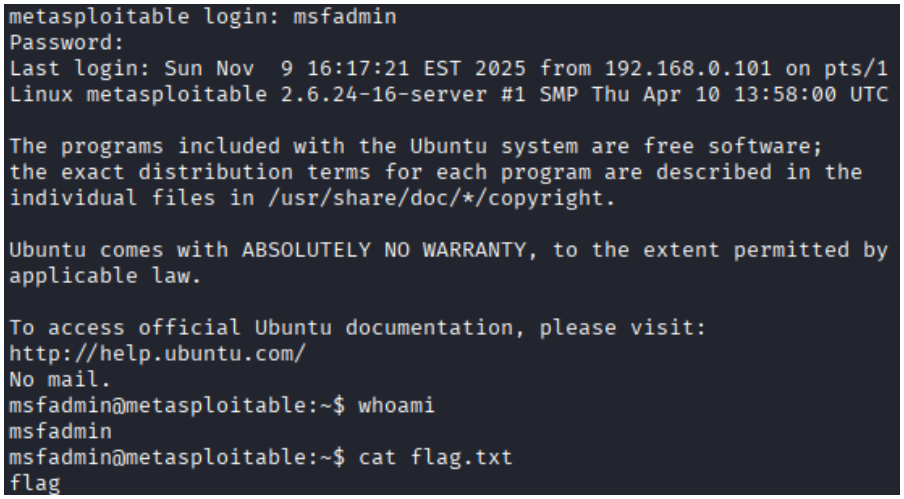


Рисунок 3.20 – Процес пост-експлуатації за допомогою вразливого telnet

3.3 Аналіз результатів тестування

Під час процесу пентестнгу було протестовано три вразливості: працюючий VNC-сервер, вразливий vsftpd та конфігураційний недолік під час підключення до telnet.

Експлуатація за допомогою Metasploit Framework продемонструвала можливість підключення до VNC. Основними проблемами є: незашифрований трафік, слабка аутентифікація, яка була пройдена за допомогою сканера та

можливість доступу до сесії десктопу. Ризик вразливості - високий. Якщо мережа не сегментована - атакуючий отримує найбільший рівень прав в системі та контроль над цілою інфраструктурою.

Також, під час сканування було виявлено службу FTP з банером “vsftpd 2.3.4”. Використовуючи Metasploit Framework було відтворено POC, який веде до можливості виконувати команди в системі та отримання shell. Високий пріоритет вразливості полягає в тому, що експлуатація надає можливість виконувати код на сервері, а це відкриває шлях до ескалації.

Сканування показало, що на сервері є відкритий порт 23 із можливістю підключення до нього за допомогою стандартних облікових даних, які подаються у образі при першій спробі увійти в систему. Загроза полягає в тому, що будь-який учасник мережі може отримати доступ до сесії. Такий ризик - це критична вразливість: конфіденційні дані, облікові дані, дані сесій можуть бути відкриті для атакуючого. Це може створити високу ймовірність експлуатації.

3.4 Рекомендації та план усунення

У випадку із VNC-сервером існує короткострокові рішення: застосувати брандмауерні правила та закрити прямий доступ із зовнішніх мереж. Більш комплексне рішення: увімкнути шифрування, встановити більш складні паролі, обмежити кількість сесій, застосувати блокування після невдалих спроб. Також, існує ряд організаційних заходів: мінімізувати віддалені сесії та десктопи, моніторинг сесій та централізована аутентифікація.

Також, необхідно негайно видалити або зупинити службу vsftpd на цільовій машині та заблокувати порт 21 на рівні мережі, якщо FTP не потрібен. Якщо сервіс або порт необхідні, потрібно оновити програмне забезпечення до останньої безпечної версії або перейти на сучасніші рішення із TLS. Також, варто заборонити анонімний доступ, реалізувати процес логування та обмежити привілеї процесу. Серед процедурних заходів можна виділити: регулярні патч-скани та впровадження політики оновлення програмного забезпечення.

Для telnet необхідно вимкнути стандартні облікові дані, встановити складні паролі та використовувати багатофакторну аутентифікацію. Але основними рішеннями проблеми є перехід на зашифровані протоколи та централізована політика управління обліковими даними.

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі представлена практична демонстрація процесу аналізу вразливостей інформаційних систем. Було використано раніше згадані інструменти: Nmap, Nessus, Metasploit Framework та авторський інструмент для аналізу виводу сканеру за допомогою штучного інтелекту.

Аналіз базувався на лабораторному образі Metasploitable. Було знайдено вразливості, виявлено їх вплив на систему, виконано процес експлуатації та пост-експлуатації для точної верифікації вразливостей.

Здебільшого, результати обмежені контекстом тестової, віртуальної мережі. У реальному середовищі додатковими факторами будуть проміжні пристрої, політики безпеки, віртуальні приватні мережі та захищені канали та порти. Не дивлячись на неповноту картини, результати розділу - задовільні, як в навчальних цілях, так і в контексті досліджень.

Наступними кроками в тестуванні, в рамках дослідження можуть бути: перевірка та експлуатація інших портів та сервісів, за наявності, перевірка інших вузлів в мережі на наявність досліджених вразливостей та цілеспрямовані атаки на окремі версії сервісів.

ВИСНОВКИ

Метою роботи було дослідження теоретичних та практичних методів аналізу інформаційних систем за допомогою етичного хакінгу, що дає змогу ефективніше швидше та якісніше організовувати безпеку даних та інформації.

Для формування загального уявлення про предмет дослідження, на перших етапах було визначено явища інформаційних систем та вразливостей.

Інформаційна система була розглянута як набір взаємопов'язаних компонентів: апаратне та програмне забезпечення, бази даних, мережі та користувачі.

У загальному розумінні вразливості були поділені на фізичні, організаційні та технічні. За класифікацією вразливості інформаційних систем були визначені як: апаратні, програмні, мережеві, процедурні та людські. Була проведена кореляція між класифікованими вразливостями та їх джерелами виникнення. Визначено, що проблема масштабується тоді, коли масштабується система.

Ефективним методом забезпечення захисту даних та боротьби з вразливостями є процес імітації дій хакера – етичний хакінг, або пентестинг. Виділено три типи тестування на проникнення: метод чорної скриньки, метод сірої скриньки та метод білої скриньки.

Було визначено поняття стандартизації в кібербезпеці та впливу цього явища на процеси етичного хакінгу. Необхідність стандартів та нормативних баз полягає у тому, щоб пентестинг був юридично безпечним, етичним та регульованим процесом. Також, питання етики є одним з ключових при дослідженні етичного тестування на проникнення, адже будь-яка взаємодія із чутливими даними порушує питання про моральну складову цього процесу.

Наступним кроком у процесі дослідження було визначено поширені методології, які формуються на базі стандартів та нормативних баз. Також, було досліджено найпоширеніші інструменти, які використовуються у процесі етичного тестування на проникнення, а саме: Nmap, Nessus та Metasploit. Nmap – це

консольний сканер, який здатен виявляти відкриті порти, сервіси та версії програмного забезпечення на цілі. Nessus – це сканер, який має власну базу даних вразливостей і надає інструкції як використати ту чи іншу вразливість. Metasploit Framework – це спеціальна програма, яка містить вже реалізовані корисні навантаження, які можна виконати проти цілі. Перераховані інструменти, як і будь-які інструменти в етичному хакінгу не мають чітких вимог і їх вибір залежить від вимог для конкретної задачі. В локальній безпечній лабораторії було досліджено роботу інструментів для більш чіткого розуміння їхнього впливу на процес аналізу вразливостей.

В межах додаткового наукового дослідження було розглянуто вплив нового світового тренду – штучного інтелекту на процес етичного тестування на проникнення. Було виявлено, що для аналізу результатів сканування немає інструменту, який використовуючи генеративну модель штучного інтелекту, здатен генерувати чіткі рекомендації для подальших дій тестування. Разом із тим існують супутні ризики для реалізації такого підходу. Перша проблема – генеративний штучний інтелект може галюцинувати, тобто видавати відповіді, які не відповідають запиту. Друга – проблема захисту даних, адже використовуючи сторонні сервіси, існує ризик витоку даних.

Для демонстрації було розроблено невелику модульну інтеграцію із Gemini API. Програмний код, реалізований на Python, використовує готові результати сканування Nmap і відправляє їх у хмарний сервіс. Після обробки дані повертаються із конкретними порадами, що можна спробувати протестувати. Протестована програмна реалізація була у локальній лабораторії, для забезпечення безпеки та етичності процесу дослідження.

Досліджувані інструменти та програмна реалізація модульної інтеграції, для практичної демонстрації були протестовані у локальному безпечному середовищі Metasploitable. Metasploitable – це машина, на якій встановлені різні версії вразливого програмного забезпечення. Вона була створена для навчання та демонстрації процесів етичного тестування на проникнення. В якості хоста, було використано віртуальну машину Kali Linux. Були продемонстровані процеси

пошуку, експлуатації та пост-експлуатації вразливостей трьох вразливих сервісів. Наступним кроком в межах практичної демонстрації, були запропоновані плани усунення вразливостей та загальний аналіз знахідок для кожного з сервісів.

Тож, ефективний тест на проникнення – це не просто набір інструментів або формальний перелік етапів, а інтегрований підхід, який поєднує методологічну базу, технічні вміння та можливості і помірковану автоматизацію. Тобто, це процес, який в якійсь мірі, вимагає творчого підходу, враховуючи комплексну складову інформаційних систем.

Також враховуючи тенденції сьогоднішнього світу, швидкості розвитку вразливостей та загроз для інформаційних систем етичне тестування на проникнення може стати не просто методом пошуку вразливостей, а процесом аналізу стану безпеки в динаміці, який дозволяє оцінити не лише наявність конкретних технічних недоліків, а й загальну здатність системи протистояти сучасним загрозам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Закон України «Про Стратегію кібербезпеки України». [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/447/2021#

2. Уразливість в інформаційних технологіях [Електронний ресурс] / Wkipedia. — Режим доступу: https://en.wikipedia.org/wiki/Vulnerability_(computer_security)

3. A Practical Guide to Cybersecurity Risk Management (ISO/IEC 27005:2022. — [Чинний від 2022-01-01]. — International Organization for Standardization. — (Міжнародний стандарт).

4. Darril Gibson. Microsoft Networking Essentials / Darril Gibson. — Wiley Publishing, Inc. — 2011. — 371 с. — (RepoTzku)

5. Dark Web Statistics & Trends for 2025 [Електронний ресурс] / Juan H. // Prayproject. — 2025. — Режим доступу: https://preyproject.com/blog/dark-web-statistics-trends

6. Distribution of vulnerabilities by CVSS scores. Vulnerabilities by type and year [Електронний ресурс] / Cvedetails. — Режим доступу: https://www.cvedetails.com/

7. Georgia Weidman. A Hands-On Introduction to Hacking / Georgia Weidman. San Francisco. — 2014. — 339 p.

8. History of Ethical Hacking. [Електронний ресурс] / Redteamacademy. — 2024. — Режим доступу: https://redteamacademy.com/history-ofethical-hacking/

9. Information Security Risk Management (ISO/IEC 27005:2011). — [Чинний від 2011-01-01]. — International Organization for Standardization. — (Міжнародний стандарт).

10. Information Systems for Business and Beyond [Електронний ресурс] / Opentextbook. — Режим доступу: https://opentextbook.site/informationsystems2019/chapter/chapter-1- what-is-an-information-system-information-systems-introduction/

11. Information Systems Security Threats and Vulnerabilities: A case of the Institute of Accountacny Arusha // Journal of Computer and Communications. — November 2022. — 15 p.

12. Jon Erickson. Hacking: The Art of Exploitation / Jon Erickson. — No Starch Press. — 2008. — 488 p.

13. Kali Linux Documentation [Електронний ресурс] / Kali. — Режим доступу: https://www.kali.org/docs/

14. Metasploit Framework Documentation [Електронний ресурс] / Rapid7 Metasploit. — Режим доступу: https://docs.metasploit.com/

15. Nessus Documentation [Електронний ресурс] / Tenable. — Режим доступу: https://docs.tenable.com/nessus/10_10/Content/GettingStarted.htm

16. Nmap Network Scanning. The Official Nmap Project Guide to Network Discovery and Security Scanning [Електронний ресурс]. — Режим доступу: https://nmap.org/book/toc.html

17. OSSTMM. The Open Source Security Testing Methodology Manual [Електронний ресурс] / ISECOM. — 2024. — Режим доступу: https://www.isecom.org/OSSTMM.3.pdf

18. OWASP Top Ten [Електронний ресурс] / OWASP. — 2025. — Режим доступу: https://owasp.org/www-project-top-ten/

19. OWASP Web Security Testing Guide [Електронний ресурс] / OWASP. — 2025. — Режим доступу: https://owasp.org/www-projectweb-security-testing-guide/

20. Python 3 Documentation [Електронний ресурс] / Python. — Режим доступу: https://docs.python.org/3/

21. Splunk (Cisco Company). Vulnerabilities, Threats and Risk Explained [Електронний ресурс]. — Режим доступу: https://www.splunk.com/en_us/blog/learn/vulnerability-vs-threat-vsrisk.html

22. Staysafeonline. The Evolution Of Ethical Hacking. [Електронний ресурс]. — 2024. — Режим доступу: https://www.staysafeonline.org/articles/the-evolution-of-ethical-hackingfrom-curiosity-to-cybersecurity

23. The NIST Cybersecurity Framework: Guide for Conducting Risk Assessments. — [Чинний від 2024-03-26]. — National Institute of Standards and Technology. — (Міжнародний стандарт). 71

24. Vulnerability Scanning Tools [Електронний ресурс] / OWASP. — Режим доступу: https://owasp.org/wwwcommunity/Vulnerability_Scanning_Tools/

25. Vulnerabilities in Information Security. [Електронний ресурс] / GeeksForGeeks. — 2024. — Режим доступу: https://www.geeksforgeeks.org/ethical-hacking/vulnerabilities-ininformation-security/

ДОДАТОК А

Міністерство освіти і науки України
Харківський національний університет імені В. Н. Каразіна
Навчально-науковий інститут комп’ютерних наук та штучного інтелекту

Метод розробки модульного інструменту для сканування інформаційних мереж

Харків – 2025

```
from __future__ import annotations
import os
import sys
import json
import argparse
import xml.etree.ElementTree as ET
from pathlib import Path
from dotenv import load_dotenv
from rich.console import Console
import time
import random
import re

load_dotenv()
console = Console()

GEMINI_KEY = os.getenv("GEMINI_API_KEY")
DEFAULT_MODEL = os.getenv("GEMINI_MODEL", "models/gemini-2.5-flash-lite")
CHUNK_CHARS = int(os.getenv("CHUNK_CHARS", "3500"))
MAX_OUTPUT_TOKENS = int(os.getenv("MAX_OUTPUT_TOKENS", "6000"))

SYSTEM_PROMPT = (
    "you are an experienced, senior cybersecurity analyst and penetration tester. "
    "Givne network scan data, produce a highly detailed, actionable assessment. Your respoonse MUST contain "
    "TWO clearly separated parts: a long human-readable techiacal analysis followed by a camhine-parseable JSON block "
    "wrapped in ``json`` fences.\n\n"

    "PART 1 - DETAILED TECHICAL ANALYSIS (text):\n"
    "- Write a structured, multi-paragraph analysis (at least 4 paragraphs) that covers: "
    "overall security posture, prioritized findings, likely attack paths, and the practical impact of each findings. "
    "- For each importnat finding include: short title, clear description, why it matters, exploitability (Trviial/Easy/Medium/Hard), "
    "and an example of how an attacker could leverage it in the real world.\n"
    "- Provide exact, reproduciable verification/test stepts for each finding. For each step give the specific commands or tool and flags to use, "
    "expected output indicating a positive result, and any safe caveats (do not instruct destuctive testing unless explicitly asked). "
    "- Provide precise remediation guidance for each finding: configuration chages, package names and versions to upgrade to, sample commands, "
    "policy changes, OpenSSL commands, firewall rules). "
```

```
        "- For each remediation, add verification steps showing how to confirm the issue is fixed (exact commnads or tests and expected outupts).\n"
        "- Where applicable include CVE identifires and short notes about public expoloit availablity or proof-of-conept; if you reference a CVE, include its year and number. "
        "- Prioritize findings (Critical/High/Medium/Low) and include an estimated remediation effort for each (Estimated Effort: <minutes/hours/days>). "
        "- At the end of the analysis include: a concise prioritized action list (top 5 actions), a suggested timeline (Immidiate/1-7 days/2-4 weeks), "
        "and an overall confidence score (0-100%) with a one-sentence justification.\n\n"

        "PART 2 - STRUCRED JSON SUMMARY (machine-friendly):\n"
        #"- Provide a valid JSON object in a triple backtick ```json``` code block. The JSON must follow this schema exactly:\n"

        " Do not wrap it in backticks."
        "Do not add explanations."
       "Output must start with '{' and end with '}' "

        "{\n"
        '  \"summary\": \"<2-3 sentence summary of the most important issues>\", \n'
        '  \"confidence\": \"<0-100% numeric or string>\", \n'
        '  \"prioritized_actions\": [\"<top action 1>\", \"<top action 2>\", \"...\",\n'
        '  \"findings\":[\n'
        '    {\n'
        '      \"title\": \"<short finding title>\", \n'
        '      \"severity\": \"<Critical/High/Medium/Low>\", \n'
        '      \"exploitability\": \"<Trivial/Easy/Medium/Hard>\", \n'
        '      \"evidence\": \"<exact service line or scan output used as evidence>\", \n'
        '      \"cve\": \"<CVE-YYYY-NNNN or empty>\", \n'
        '      \"test_steps\": [\"<exact command or step to verify>\", \"...\"], \n'
        '      \"remediation\": \"<clear remediation steps and commands>\", \n'
        '      \"remediation_verification\": [\"<command or check to confirm fix>\", \"...\"], \n'
        '      \"estimated_effort\": \"<minutes/hours/days>\", \n'
        '      \"confidence\": \"<0-100%>\", \n'
        '    }\n'
        '   ]\n'
        " }\n\n"

        "REQUIREMENTS AND STYLE NOTES: \n"
        "- The text analysis must be detailed and professional (no casual tone). Use precise technical wording and provide actionable commands and config snippets where appropriate. "
```

"- The JSON must be sytactically valid JSON enclosed in ```json fences and parsable by standard JSON parsers. Do NOT include any etra prose inside the JSON block. "
"- If you cannot find any issues, still produce a thorough textual explanation describing why the scan appears clean and what additional tests coudl be run. "
"- Always prefer safe, label it clearly as intusive and include safety/warning notes. "
"- When giving commands, prefer cross-Unix examples (bash) and also provide common Widnows equivalents if relevant for the finding."
)

```
def read_file(path: str) -> str:
    if not path or not os.path.exists(path):
        return ""
    with open(path, "r", encodnig="utf-8", errors="ignore") as f:
        return f.read()

def parse_nmap_xml(path: str) -> str:
    if not path or not os.path.exists(path):
        return ""
    try:
        tree = ET.parse(path)
        root = tree.getroot()
    except Exception as e:
        console.print(f"[red]Failed to parse {path}: {e}[/red]")
        return ""
    rows = []
    for host in root.findall(".//host"):
        addr_el = host.find("./address")
        ip = addr_el.get("addr") if addr_el is not None else ""
        hostnames = [hn.get("name") for hn in host.findall("./hostnames/hostname")] or []
        for port in host.findall(".//ports/port"):
                pnum = port.get("portid", "")
                proto = port.get("protocol", "")
                state_el = port.find("state")
                state = state_el.get("state") if state_el is not None else ""
                svc = port.find("service")
                svc_name = svc.get("name") if svc is not None else ""
                svc_prod = svc.get("product") if svc is not None else ""
                svc_ver = svc.get("version") if svc is not None else ""
                extrainfo = svc.get("extrainfo") if svc is not None else ""
                host_display = ip or (hostnames[0] if hostnames else "unknown")
                line = f"{host_display} {pnum}/{proto} {state} {svc_name} {svc_prod} {svc_ver} {extrainfo}".strip()
                rows.append(line)
```

```
    return "\n".join(rows)


def chunk_text(s: str, max_chars: int = CHUNK_CHARS):
    if not s:
        return []
    return [s[i:i + max_chars] for i in range(0, len(s), max_chars)]


def init_genai(key: str):
    if not key:
        raise SsytemExit("GEMINI_API_KEY not found in environment or .env")
    try:
        import google.generativeai as genai
    except Exception as e:
        raise SystemExit(f"Missing dependency google-generativeai: {e}")
    genai.configure(api_key=key)
    return genai

def call_gemini_once(genai, prompt: str, model: str = DEFAULT_MODEL,
max_output_tokens: init = MAX_OUTPUT_TOKENS):
    try:
        model_obj = genai.GenerativeModel(model)
        response = model_obj.generate_content(
                prompt,
                generation_config={
                        "max_output_tokens": max_output_tokens,
                        "temperature": 0.0,
                },
        )
        raw = getattr(response, "text", None)
        if raw is None:
                raw = str(response)
        return raw
    except Exception as e:
        raise RuntimeError(f"Gemini call failed: {e}")

def call_with_retry(genai, prompt: str, model: str = DEFAULT_MODEL,
max_output_tokens: int = MAX_OUTPUT_TOKENS, max_retries: int = 4, base_delay:
float = 1.0):
    for attemtp in range(1, max_retries + 1):
        try:
                return call_gemini_once(genai, prompt, model=model,
max_output_tokens=max_output_tokens)
        except RuntimeError as e:
```

```
                msg = str(e).lower()
                if any(token in msg for token in ("429", "resource_exhausted", "too many
requests", "quota")):
                        if attempt == max_retries:
                                raise
                        sleep = base_delay * (2 ** (attempt - 1)) + random.uniform(0, 0.5)
                        console.print(f"[yellow]Rate linit detected - retry
{attempt}/{max_retries} after {sleep:.1f}s[/yellow]")
                        time.sleep(sleep)
                        continue
                else:
                        raise

def try_load_json(s: str):
    try:
        return json.loads(s)
    except Exception:
        return None


def repari_common_issues(s: str) -> str:
    s = re.sub(r"^\s*```(?:json)?\s*", "", s, flags=re.IGNORECASE)
    s = re.sub(r"s*```\s*$", "", s, flags=re.IGNORECASE)
    if "\\n" in s or "\\t" in s or '\\"' in s:
        try:
                s = bytes(s, "utf-8").decode("unicode_escape")
        except Exception:
                s = s.replace("\\n", "\n").replace("\\t", "\t").replace('\\"', '"').replace("\\\\",
"\\")
    s = re.sub(r"[x00-x08\x0b\x0c\x0e-\x1f]", "",s)
    s = re.sub(r",\s*([}\]])", r"\1", s)
    s = re.sub(r"}\s*{", "}, \n{", s)
    s = re.sub(r",\s*{", "}, \n{", s)

def extract_json_from_text(s: str):
    if not s:
        return None
    s = s.strip()
    blocks = re.findall(r"```json\s*(.*?)\s*```", s, flags=re.DOTALL | re.IGNORECASE)
    if blocks:
        for block in blocks:
                parsed = try_load_json(block)
                if parsed:
                        return parsed
        s = "\n".join(blocks)
```

```
    s = re.sub(r"^\s*```(?:json)?\\n?", "", s, flags=re.IGNORECASE)
    s = re.sub(r"```$", "", s.strip(), flags=re.IGNORECASE)
    parsed = try_load_json(s)
    if parsed:
        return parsed
    parsed = try_load_json(s)


def generate_markdown_report(parsed_json: dict | None, outfile: str, meta: dict = None,
analysis_text: str = ""):
    lines = []
    meta = meta or {}
    lines.append("# AI Recon Report\n")

    if meta.get("target"):
        lines.append(f"**Target:** {meta['target']}\n")

    if analysis_text:
        lines.append("## Detailed Analysis\n")
        lines.append(analysis_text.strip() + "\n")

    if not parsed_json:
        lines.append("_Structured JSON summary not available_")
    else:
        lines.append("## Summary\n")
        lines.append(parsed_json.get("summary", "No summary proviede.") + "\n")

        lines.append("## Findings\n")
        findings = parsed_json.get("findings", [])
        if not findings:
                lines.append("_No findings reported._\n")
        else:
                for i, f in enumerate(findings, 1):
                        title = f.get("title", "Unitled")
                        sev = f.get("severity", "Unknown")
                        lines.append(f"### {i} {title} - {sev}\n")

                        evidence = f.get("evidnce", "")
                        if evidence:
                                lines.append(f"**Evidence:**\n> {evidence}\n")

                        steps = f.get("steps_to_verify", [])
                        if steps:
                                lines.append("**steps_to_verify", [])
                                for st in steps:
```

```
                                lines.append(f"- `{st}`")

                        rem = f.get("remediation", "")
                        if rem:
                                lines.append(f"**Remediation:**\n{rem}\n")

        content = "\n".join(lines)
        Path(outfile).parent.mkdir(parents=True, exist_ok=True)
        with open(outfile, "w", encoding="utf-8") as f:
                f.write(content)
        return outfile


def main():
    parser = argparse.ArgumentParser(description="AI Recon Assistant (Gemini V)")
    parser.add_argument("--nmap", default="scan.xml", help="Nmap XML file path")
    parser.add_argument("--nikto", default="nikto.txt", help="Nikto output file path (its
optional)")
    parser.add_argument("--out", default="reports/report.md", help="Outpu Markdonw
report")
    parser.add_argument("--model", default=DEFAULT_MODEL, help="Gemini model
(default: gemini-2.5-flash-lite)")
    args = parser.parse_args()

    console.rule("[bold green]AI Recon Assistant[/]")
    nmap_text = parse_nmap_xml(args.nmap)
    nikto_text = read_file(args.nikto)
    combined = f"NMAP:\n{nmap_text}\n\nNIKTO:\n{nikto_text}"
    console.print(f"[blue]Prepared payload - {len(combined)} characters[/blue]")

    genai = init_genai(GEMINI_KEY)
    chunks = chunk_text(combined, CHUNK_CHARS)
    responses = []
    Path("reports").mkdir(parents=True, exist_ok=True)

    for idx, chunk in enumerate(chunks, start=1):
        prompt = SYSTEM_PROMPT + "\n\nUser data:\n" + chunk
        console.print(f"[yellow]Sending chunk {idx}/{len(chunks)}..[/yellow]")
        try:
                raw = call_with_retry(genai, prompt, model=args.model,
max_output_tokens=MAX_OUTPUT_TOKENS)
        except Exception as e:
                console.print(f"[red]Error contacting Gemini: {e}[/red]")
                sys.exit(2)
```

```
        raw_part = ""
        match = re.split(r"```json", raw, maxsplit=1, flags=re.IGNORECASE)
        text_part = match[0].strip() if len(match) > 1 else raw.strip()
        parsed = extract_json_from_text(raw)

        if parsed is None:
                text_path = Path("reports") / f"text_analysis_chunk{idx}.txt"
                with open(text_path, "w", encoding="utf-8") as tf:
                        tf.write(text_part)
                console.print(f"[yellow]JSON not detected, saved tet analysis ->
{text_path}[/yellow]")
                responses.append({"text": text_part, "json": None})
        else:
                with open(Path("reports") / f"parsed_response_chunk{idx}.json", "w",
encoding="utf-8" ) as pf:
                        json.dump(parsed, pf, ensure_ascii=False, indent=2)
                with open(Path("reports") / f"text_aanlysis_chunk{idx}.txt", "w", encoding
= "utf-8")as tf:
                        tf.write(text_part)
                responses.append({"text": text_part, "json": parsed})

    if not responses:
        console.print("[red]NO response received from model.[/red]")
        sys.exit(3)
    merged = responses[0]
    analysis_text = merged.get("text", "")
    parsed_json = merged.get("json", None)
    md_path = generate_markdown_report(parsed_json, args.out,
analysis_text=analysis_text)

    console.print(f"[green]Generated Markdown report: {md_path}[/green]")
    console.print("[bold blue]Done.[/bold blue]")

if __name__ == "__main__":
        main()
```