МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

Інститут комп’ютерних систем
Кафедра комп’ютеризовані системи та програмні технології

Бобок І.І.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни
«СУЧАСНІ МЕТОДИ ЗАХИСТУ КОМП’ЮТЕРНИХ МЕРЕЖ»
(9 семестр)
для здобувачів освітньо-кваліфікаційного рівня «Магістр»
за спеціальністю F5 Кібербезпека та захист інформації

Одеса – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»

Інститут комп’ютерних систем
Кафедра комп’ютеризовані системи та програмні технології

Бобок І.І.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

з дисципліни
«СУЧАСНІ МЕТОДИ ЗАХИСТУ КОМП’ЮТЕРНИХ МЕРЕЖ»
(9 семестр)
для здобувачів освітньо-кваліфікаційного рівня «Магістр»
за спеціальністю F5 Кібербезпека та захист інформації

Затверджено
на засіданні кафедри
комп’ютеризовані системи та
програмні технології
Протокол №__ від _____

Одеса – 2026

Методичні рекомендації до виконання лабораторних робіт з дисципліни «Сучасні методи захисту комп’ютерних мереж» (9 семестр) для здобувачів освітньо-кваліфікаційного рівня «Магістр» за спеціальністю F5 Кібербезпека та захист інформації / Укл.: І.І. Бобок. — Одеса: Національний університет «Одеська політехніка», 2026. — 117 с.

Укладач: Бобок І.І., д.т.н., професор

ЗМІСТ

АНОТАЦІЯ

Предметом вивчення дисципліни «Сучасні методи захисту комп'ютерних мереж» є принципи, технології та інструменти забезпечення цілісності, конфіденційності та доступності мережевої інфраструктури в умовах сучасних кіберзагроз, з акцентом на архітектурні підходи, автоматизацію захисту та хмарно-орієнтовані рішення.

Метою дисципліни є формування комплексу знань, пов'язаних із вирішенням задач захисту соціотехнічної системи, й специфіку реалізації інформаційних операцій та атак в соціотехнічних системах; в дисципліні приділено значну увагу типовим інцидентам у сфері високих технологій, а також методам і засобам соціального інжинірингу, що забезпечує набуття здобувачами вищої освіти узагальненого знання щодо заходів із захисту від соціотехнічних атак та поширює їх в напрямку основ професійної діяльності.

Завдання вивчення дисципліни:

- знати основи системного методу вирішення задач захисту соціотехнічної системи;
- здійснювати прогнозування поведінки соціотехнічної системи;
- знати специфіку реалізації інформаційних операцій та атак в соціотехнічних системах;
- освоїти технології, техніки та механізми забезпечення інформаційної безпеки соціотехнічних систем, виявлення і протидії соціотехнічним атакам та впливу на колективну та індивідуальну свідомість;
- вміти провести заходи по плануванню інформаційних операцій та атак;
- вміти керувати соціотехнічними системами в контексті необхідності забезпечення їх інформаційної безпеки.

Таким чином, стратегічні цілі дисципліни полягають у націлені майбутніх фахівців на творче застосування, розвиток, удосконалення отриманих знань у подальшій професійній підготовці та їх наступній практичній діяльності.

Метою дисципліни «Сучасні методи захисту комп'ютерних мереж» є розвиток теоретичних знань та практичних навичок із побудови, керування, модернізації, моніторингу та аналізу продуктивності, діагностики та розв`язання проблем забезпечення безпеки сучасних комп'ютерних мереж; формування у майбутніх фахівців умінь та компетентностей для забезпечення ефективного захисту інформації; оволодіння майбутніми фахівцями сучасними методами та засобами захисту інформації в умовах широкого використання сучасних інформаційних технологій.

Завдання вивчення дисципліни:

— сформувати навички здобувачів вищої освіти визначати види і можливі методи та шляхи реалізації загроз безпеці комп'ютерних мереж на основі аналізу структури та змісту інформаційних процесів підприємства; визначати прояви мережевих атак; обґрунтовувати організаційно-технічні заходи щодо захисту інформації в комп'ютерних мережах;

— розвинути навички здобувачів вищої освіти виконувати роботи по установці, налаштуванні і обслуговуванню технічних та програмно-апаратних засобів захисту інфраструктури комп'ютерних мереж;

— сформувати навички розробляти комплекс заходів (правила, процедури, практичні прийоми і ін.) для управління інформаційною безпекою.

Стратегічні цілі дисципліни – націлити майбутніх фахівців на творче застосування, розвиток, удосконалення отриманих знань у подальшій професійній підготовці та їх наступній практичній діяльності.

Програмні результати навчання:

ПРН6. Аналізувати та оцінювати захищеність систем, комплексів та засобів кіберзахисту, технології створення та використання спеціалізованого програмного забезпечення.

ПРН8. Досліджувати, розробляти і супроводжувати системи та засоби інформаційної безпеки та/або кібербезпеки на об'єктах інформаційної

діяльності та критичної інфраструктури.

ПРН12. Досліджувати, розробляти та впроваджувати методи і заходи протидії кіберінцидентам, здійснювати процедури управління, контролю та розслідування, а також надавати рекомендації щодо попередження та аналізу кіберінцидентів в цілому.

ЛАБОРАТОРНА РОБОТА 1: Перехід від периметрАЛЬНОї моделі мережевої безпеки до архітектури Zero Trust

Мета: ознайомитися з обмеженнями периметральної моделі мережевої безпеки та практично реалізувати перехід до архітектури Zero Trust у два послідовні етапи на реальних контейнеризованих сервісах.

Теоретична частина

Традиційний підхід до мережевої безпеки протягом багатьох років базувався на периметральній моделі, яку часто порівнюють із концепцією «замку та рову» (*Castle-and-Moat*). Основна ідея полягала у створенні міцного захисного бар’єру на межі корпоративної мережі за допомогою брандмауерів, систем виявлення вторгнень (IDS/IPS) та VPN. Усередині цього периметра користувачі, пристрої та додатки вважалися апріорі довіреними. Такий підхід був ефективним у часи, коли всі корпоративні ресурси фізично знаходилися в локальних центрах обробки даних, а співробітники працювали виключно з офісних комп'ютерів.

Однак стрімкий розвиток інформаційних технологій та еволюція кіберзагроз кардинально змінили ландшафт безпеки. Масова міграція сервісів у хмару, поширення віддаленої роботи, використання особистих пристроїв (BYOD) та постійна інтеграція з мережами підрядників призвели до того, що класичний корпоративний периметр фактично розмився або зовсім зник.

Змінився і спектр мережевих загроз. Сучасні атаки, зокрема складні цілеспрямовані загрози (APT), атаки на ланцюжки постачання (*Supply Chain Attacks*) та масове використання скомпрометованих облікових даних, довели критичну вразливість периметральної моделі. Якщо зловмиснику вдається подолати зовнішній захист або ж загроза виникає зсередини (інсайдер чи заражений пристрій співробітника), він отримує безперешкодний доступ до широкого пулу внутрішніх ресурсів. Ця надмірна довіра всередині мережі створює ідеальні умови для горизонтального переміщення (*lateral movement*),

дозволяючи хакерам непомітно підвищувати привілеї та отримувати доступ до критично важливих даних.

У відповідь на ці виклики індустрія кібербезпеки перейшла до нової парадигми — архітектури нульової довіри ZTA *(Zero Trust Architecture*). Фундаментальний принцип цієї моделі звучить як «ніколи не довіряй, завжди перевіряй» (*Never trust, always verify*). Згідно з концепцією Zero Trust, довіра не надається автоматично на основі мережевого розташування (наприклад, самого факту підключення до корпоративного Wi-Fi або VPN). Замість цього кожен запит на доступ розглядається як потенційно ворожий, незалежно від того, звідки він надходить – ззовні чи зсередини мережі.

Впровадження ZTA вимагає перенесення фокусу захисту з широких мережевих сегментів на конкретні дані, додатки та ідентичності. Це досягається завдяки:

— Мікросегментації: поділу мережі на дрібні, ізольовані зони для стримування потенційних зломів.
— Безперервній автентифікації та авторизації: перевірці не лише пароля, але й контексту запиту (геолокації, часу, поведінкових аномалій).
— Оцінці стану пристроїв: доступу лише з комп'ютерів, що відповідають корпоративним політикам безпеки (наявність оновлень, працюючий EDR).
— Принципу мінімальних привілеїв (*Least Privilege*): наданню доступу виключно до тих ресурсів, які необхідні для виконання конкретного завдання «тут і зараз».

Ця лабораторна робота присвячена практичному вивченню архітектури Zero Trust та її ключових відмінностей від класичної периметральної моделі.

Важливе уточнення щодо обраного підходу: мікросегментація в цій роботі реалізована на основі ізольованих Docker-мереж – це host/workload-based підхід (той самий принцип, що лежить в основі промислових рішень на кшталт VMware NSX чи Illumio), а не мережево-апаратний підхід на основі VLAN. Головна практична відмінність

полягає в тому, що ізоляція тут забезпечується не правилами фільтрації (ACL), а фізичною відсутністю спільного мережевого простору – вузли з різних, не пов’язаних Docker-мереж не бачать одне одного навіть на рівні розв'язання імені (DNS), не кажучи вже про встановлення з’єднання.

Слід зауважити, що сама лише мікросегментація відповідає на питання «звідки» (з якого сегмента прийшов запит) і нічого не каже про «хто» надсилає запит. Якщо в сегмент, що має дозволений доступ, потрапить сторонній чи скомпрометований вузол – мікросегментація його не зупинить. Рівень ідентичності, який буде відтворений за допомогою mTLS (mutual TLS) закриває цю прогалину: на відміну від звичайного одностороннього TLS (де сертифікат сервера перевіряє лише клієнт), mTLS вимагає, щоб і клієнт пред’явив сертифікат, підписаний довіреним центром сертифікації (CA), тобто сервер перевіряє клієнта так само суворо, як клієнт перевіряє сервер.

Рис. 1.1. Схема налаштувань лабораторної роботи

Практична частина

Робота виконується на одному Windows-хості з установленим Docker Desktop.

Частина А. Побудова мережі за периметральною моделлю

1. Створити мережу й підготувати структуру проєкту:

```
docker network create corp-net
```

2. Підготувати образ web (nginx, публічний сервіс):

2.1. Створити файл index.html з простою сторінкою

```
<h1>Corporate Web Portal</h1><p>Status: OK</p>
```

2.2. Створити Dockerfile на основі nginx:alpine

```
FROM nginx:alpine
COPY index.html /usr/share/nginx/html/index.html
```

3. Підготувати образ app (внутрішній API – Flask, маршрут /api/data, що повертає JSON-відповідь):

3.1. Створити файл app.py

```
from flask import Flask
app = Flask(__name__)

@app.route("/api/data")
def data():
    return {"service": "app", "message": "internal business
logic response"}

app.run(host="0.0.0.0", port=5000)
```

3.2. Створити Dockerfile

```
FROM python:3.12-slim
RUN pip install flask
COPY app.py .
CMD ["python", "app.py"]
```

4. Підготувати образ admin (Alpine з curl/postgresql-client/bind-tools, команда `sleep infinity` для утримання контейнера активним):

4.1. Створити Dockerfile

```
FROM alpine
RUN apk add --no-cache curl postgresql-client bind-tools
CMD ["sleep", "infinity"]
```

5. Побудувати всі три образи:

```
docker build -t lab1-web C:\LABs\1\web
docker build -t lab1-app C:\LABs\1\app
docker build -t lab1-admin C:\LABs\1\admin
```

6. Підняти всі чотири контейнери (db – офіційний образ postgres) в єдиній мережі corp-net:

```
docker run -d --name web --network corp-net lab1-web
docker run -d --name app --network corp-net lab1-app
docker run -d --name db --network corp-net -e
POSTGRES_PASSWORD=demopass -e POSTGRES_DB=corpdb
postgres:16-alpine
docker run -d --name admin --network corp-net lab1-admin
```

7. Побудувати й підняти edge (nginx reverse-proxy) як єдину точку входу ззовні, що проксіює лише до web:

7.1. Створити файл nginx.conf

```
events {}
http {
    server {
        listen 80;
        location / {
            proxy_pass http://web:80;
        }
    }
}
```

7.2. Створити Dockerfile

```
FROM nginx:alpine
COPY nginx.conf /etc/nginx/nginx.conf
```

7.3. Запустити образ edge

```
docker build -t lab1-edge C:\LABs\1\edge
docker run -d --name edge --network corp-net -p 8080:80
lab1-edge
```

8. Перевірити периметр ззовні:

```
Invoke-RestMethod http://localhost:8080
```

9. Продемонструвати вразливість периметральної моделі: припустивши, що web скомпрометовано, перевірити вільний доступ зсередини мережі до решти сервісів в обхід будь-якої логічної ієрархії.

```
docker exec web curl -s http://app:5000/api/data
docker exec web nc -zv db 5432
```

```
docker exec web ping -c 2 admin
```

Усі три команди виконуються успішно (рис. 1.2): web вільно звертається і до app, і до db (в обхід app), і до admin – периметр (edge) контролює лише north-south трафік, east-west лишається повністю відкритим.

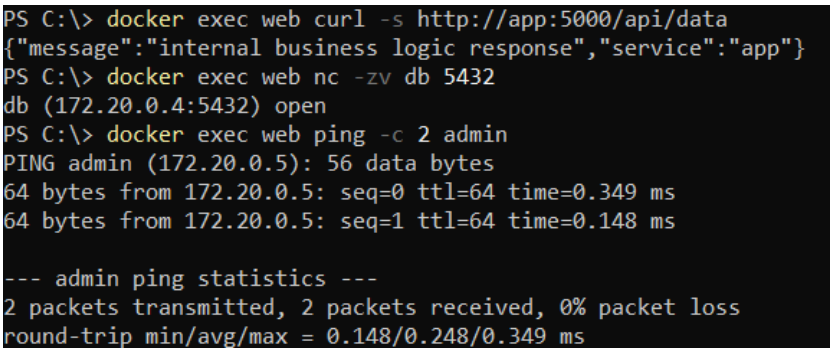


Рис. 1.2. Підтвердження вразливості периметральної моделі

Частина Б. Мікросегментація

1. Створити чотири ізольовані мережі за функціональним призначенням:

```
docker network create web-net
docker network create app-net
docker network create db-net
docker network create admin-net
```

2. Перепідключити кожен контейнер лише до необхідних мереж за принципом найменших привілеїв: web – web-net і app-net; app – app-net і db-net; db – лише db-net; admin – усі три (web-net, app-net, db-net) для адміністративного доступу; edge – переводиться на web-net (логічно належить до сегмента web).

```
docker network disconnect corp-net web
docker network connect web-net web
docker network connect app-net web

docker network disconnect corp-net app
docker network connect app-net app
docker network connect db-net app

docker network disconnect corp-net db
```

```
docker network connect db-net db

docker network disconnect corp-net admin
docker network connect web-net admin
docker network connect app-net admin
docker network connect db-net admin

docker network disconnect corp-net edge
docker network connect web-net edge
```

3. Прибрати стару «плоску» мережу

```
docker network rm corp-net
```

4. Зробити перевірку, що периметр і далі працює після переключення:

```
Invoke-RestMethod http://localhost:8080
```

Очікуваний результат, як і раніше: `<h1>Corporate Web Portal</h1>`.

Зауваження: перевірка периметра після перепідключення (`Invoke-RestMethod http://localhost:8080`) може зависнути чи повернути «connection was closed unexpectedly», що може пояснюватися тим, що nginx усередині edge розв'язав ім'я web один раз під час первинного старту (ще в мережі corp-net) і не оновлює цю адресу автоматично при зміні мережевого оточення контейнера «на льоту». Проста команда `docker restart edge` теж може не допомогти й повернути помилку «network corp-net not found», оскільки Docker зберігає початковий режим мережі (NetworkMode), вказаний при `docker run`, окремо від фактичного списку підключених мереж, і намагається відновити саме його при restart. Найнадійніше рішення в такій ситуації – не продовжувати «лагодити» контейнер у суперечливому стані, а перестворити його одразу з правильною мережею:

```
docker rm -f edge
docker run -d --name edge --network web-net -p 8080:80 lab1-edge
```

Потім повторити пеервірку

```
Invoke-RestMethod http://localhost:8080
```

5. Перевірити налаштовану мікросегментацію:

```
docker exec web curl -s http://app:5000/api/data
docker exec web nc -zv db 5432
docker exec app python3 -c "import socket;
s=socket.create_connection(('db',5432),timeout=3);
print('CONNECTED'); s.close()"
docker exec admin ping -c 2 db
```

Зауваження: команда nc відсутня в мінімальному образі python:3.12-slim, на якому побудований app (помилка «executable file not found»), тому перевірку TCP-з'єднання виконано напряму через Python (socket.create_connection) – це не позначає жодної проблеми безпеки, а лише відсутність стороннього інструменту в образі.

Отриманий результат дмонструє (рис. 1.3), що web → app успішно; web → db завершується помилкою «bad address» – DNS-сервер Docker взагалі не видає запис про db контейнеру, що не перебуває в спільній мережі db-net (тобто ізоляція діє вже на рівні видимості імені, а не лише блокування порту); app → db і admin → db – успішно (CONNECTED, ping проходить). Периметральна вразливість повністю усунена мікросегментацією, без жодного ACL чи фаєрвола, виключно топологією мереж.

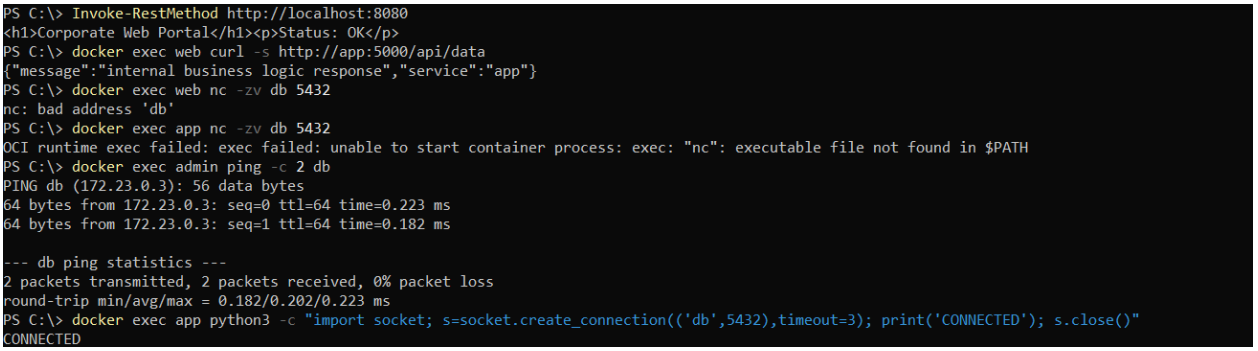


Рис. 1.3. Перевірка ефективності мікросегментації

Частина B. Рівень ідентичності (mTLS)

1. Згенерувати власний центр сертифікації (CA), сертифікат сервера (CN=db) і сертифікат клієнта (CN=postgres, збігається з іменем користувача БД,

оскільки метод автентифікації cert у PostgreSQL звіряє CN сертифіката саме з іменем користувача) – одноразовим контейнером з OpenSSL:

```
mkdir C:\LABs\1\db-ssl\certs
docker run --rm -v C:\LABs\1\db-ssl\certs:/certs alpine sh -c
"apk add --no-cache openssl >/dev/null && cd /certs && openssl
req -x509 -newkey rsa:2048 -days 365 -nodes -keyout ca.key -out
ca.crt -subj '/CN=LabCA' && openssl req -newkey rsa:2048 -nodes
-keyout server.key -out server.csr -subj '/CN=db' && openssl
x509 -req -in server.csr -CA ca.crt -CAkey ca.key
-CAcreateserial -out server.crt -days 365 && openssl req -newkey
rsa:2048 -nodes -keyout client.key -out client.csr -subj
'/CN=postgres' && openssl x509 -req -in client.csr -CA ca.crt
-CAkey ca.key -CAcreateserial -out client.crt -days 365 && ls
-la"
```

2. Побудувати кастомний образ db, що інтегрує сертифікати всередину образу (а не монтує їх з Windows напряму – bind-mount із Windows на Linux не передає коректно Unix-права доступу до приватного ключа, через що PostgreSQL відмовляється стартувати) і вмикає SSL з обов'язковим клієнтським сертифікатом (hostssl ... cert у pg_hba.conf) через init-скрипт:

2.1. Створити init-скрипт

```
#!/bin/bash
set -e
cat >> "$PGDATA/postgresql.conf" <<EOF
ssl = on
ssl_cert_file = '/certs/server.crt'
ssl_key_file = '/certs/server.key'
ssl_ca_file = '/certs/ca.crt'
EOF
cat > "$PGDATA/pg_hba.conf" <<EOF
local all all trust
hostssl all all 0.0.0.0/0 cert
hostssl all all ::0/0 cert
EOF
```

Зауваження: файл має створюватися напряму через `[System.IO.File]::WriteAllText` з явним символом `\n`, без `CRLF`, і підтверджено перевіркою `(Get-Content -Raw "C:\LABs\1\db-ssl\init-ssl.sh") -match "`r"` (має повернути `False`).

```
$content = "#!/bin/bash`nset -e`ncat >> `"`$PGDATA/postgresql.conf`" <<EOF`nssl = on`nssl_cert_file = '/certs/server.crt'`nssl_key_file = '/certs/server.key'`nssl_ca_file = '/certs/ca.crt'`nEOF`ncat > `"`$PGDATA/pg_hba.conf`" <<EOF`nlocal all all trust`nhostssl all all 0.0.0.0/0 cert`nhostssl all all ::0/0 cert`nEOF`n"
[System.IO.File]::WriteAllText("C:\LABs\1\db-ssl\init-ssl.sh", $content)
```

2.2. Створити Dockerfile

```
FROM postgres:16-alpine
COPY certs/ca.crt certs/server.crt certs/server.key /certs/
RUN chown postgres:postgres /certs/server.key /certs/server.crt /certs/ca.crt && chmod 600 /certs/server.key
COPY init-ssl.sh /docker-entrypoint-initdb.d/init-ssl.sh
RUN chmod +x /docker-entrypoint-initdb.d/init-ssl.sh
```

2.3. Побудувати і запустити образ

```
docker build -t lab1-db C:\LABs\1\db-ssl
docker stop db
docker rm db
docker run -d --name db --network db-net -e POSTGRES_PASSWORD=demopass -e POSTGRES_DB=corpdb -e POSTGRES_USER=postgres lab1-db
```

3. Перевірити, що SSL увімкнено:

```
docker exec db psql -U postgres -c "SHOW ssl;"
```

4. Інтегрувати клієнтські сертифікати в app і перевірити легітимне підключення з mTLS:

```
docker cp C:\LABs\1\db-ssl\certs\client.crt app:/tmp/client.crt
docker cp C:\LABs\1\db-ssl\certs\client.key app:/tmp/client.key
docker cp C:\LABs\1\db-ssl\certs\ca.crt app:/tmp/ca.crt
docker exec app chmod 600 /tmp/client.key
```

```
docker exec -u root app sh -c "apt-get update -qq && apt-get
install -y -qq postgresql-client >/dev/null"
```

Зауваження: встановлення пакета в неінтерактивному Debian-контейнері (образ app побудований на python:3.12-slim) супроводжується попередженнями debconf про неможливість ініціалізувати графічний/консольний фронтенд (Dialog, Readline, Teletype), що є очікуваною поведінкою за відсутності tty; встановлення завершується успішно через автоматичний перехід на фронтенд Noninteractive.

```
docker exec app psql "host=db port=5432 dbname=corpdb
user=postgres sslmode=verify-full sslrootcert=/tmp/ca.crt
sslcert=/tmp/client.crt sslkey=/tmp/client.key" -c "SELECT
'AUTHORIZED via mTLS' as result;"
```

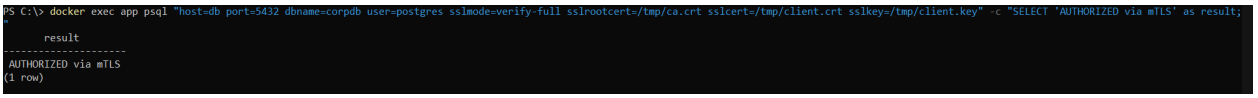


Рис. 1.4. Демонстрація легітимного доступу

5. Перевірка нелегітимного доступу з контейнера без сертифіката в тій самій мережі:

```
docker run --rm --network db-net postgres:16-alpine psql
"host=db port=5432 dbname=corpdb user=postgres sslmode=require"
-c "SELECT 1;"
```

Результат: «**FATAL: connection requires a valid client certificate**» попри повну мережеву доступність (той самий сегмент db-net) – відсутність сертифіката блокує з’єднання.

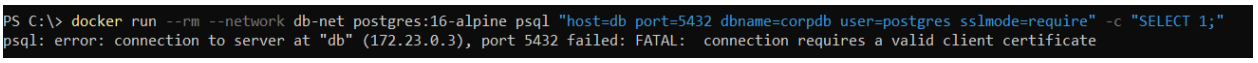


Рис. 1.5. Демонстрація нелегітимного доступу

6. Додатково можна продемонструвати відмову від обробки для підробленого (самопідписаного) сертифіката з тим самим CN, що й легітимний клієнт:

```
docker run --rm -v C:\LABs\1\db-ssl\certs:/certs --network
db-net alpine sh -c "apk add --no-cache openssl
postgresql-client >/dev/null && openssl req -x509 -newkey
rsa:2048 -days 1 -nodes -keyout /tmp/fake.key -out /tmp/fake.crt
-subj '/CN=postgres' && psql 'host=db port=5432 dbname=corpdb
user=postgres sslmode=require sslcert=/tmp/fake.crt
sslkey=/tmp/fake.key' -c 'SELECT 1;'"
```

Результат: «**SSL error: tlsv1 alert unknown ca**» – сервер відхиляє з'єднання ще на етапі TLS-рукостискання через недовіру до підпису сертифіката, раніше, ніж встиг би перевірити збіг CN з іменем користувача, що підтверджує головний висновок роботи: мережева присутність у правильному сегменті – необхідна, але не достатня умова довіри; лише підтверджена, криптографічно довірена ідентичність дає остаточну гарантію.

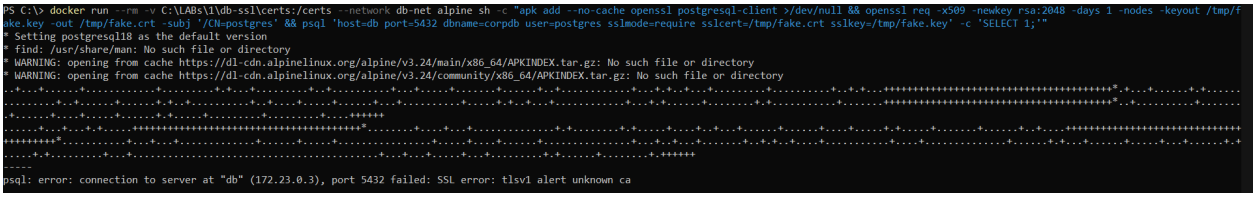


Рис. 1.6. Результат з'єднання з використанням підробленого сертифікату

Індивідуальні завдання

Кожен здобувач отримує індивідуальний варіант (табл. 1.1) власний сценарій підприємства з критичним активом і матрицею дозволених мережевих потоків. Завдання:

— побудувати власні образи (мінімум web, app, db, admin; назви й вміст сервісів адаптувати під сценарій);

— відтворити частину А (периметр, демонстрація вразливості) і частину Б (мікросегментація, усунення вразливості) для свого сценарію;

— додатково реалізувати mTLS для потоку web → app (а не лише app → db, як у базовому прикладі), самостійно визначивши, який центр сертифікації і яка пара сертифікатів для цього потрібні;

— оформити звіт зі скріншотами кожного етапу.

Таблиця 1.1. Варіанти індивідуальних завдань

№	Підприємство	Критичний актив	Дозволені мережеві потоки (Docker networks)	Потік, що демонструє вразливість периметра
1	Інтернет-магазин	payment-db (дані карток)	web-net: web, edge app-net: web, app db-net: app, payment-db, admin	web → payment-db напряму
2	Університетський портал	grades-db (успішність)	web-net: portal, edge app-net: portal, deanery-app db-net: deanery-app, grades-db, admin	portal → grades-db напряму
3	Медична клініка	patient-db (медичні картки)	web-net: kiosk, edge app-net: kiosk, ehr-app db-net: ehr-app, patient-db, admin	kiosk → patient-db напряму
4	Банківське відділення	customer-db (дані клієнтів)	web-net: teller-ws, edge app-net: teller-ws, core-banking-app db-net: core-banking-app, customer-db, admin	teller-ws → customer-db напряму
5	Промислове підприємство (IT/OT)	historian-db (архів даних)	web-net: corporate-it, edge app-net: corporate-it, scada-gateway db-net: scada-gateway, historian-db, admin	corporate-it → historian-db напряму
6	Державна установа	citizens-db (персональні дані)	web-net: public-portal, edge app-net: public-portal, case-mgmt-app db-net: case-mgmt-app, citizens-db, admin	public-portal → citizens-db напряму

Контрольні питання

1. У чому принципова відмінність периметральної моделі мережевої безпеки від архітектури Zero Trust?
2. Чому периметральна модель безсила проти загрози, яка вже отримала доступ до внутрішньої мережі (east-west трафік)?
3. Чим мікросегментація на основі ізольованих Docker-мереж (host/workload-based) відрізняється від мікросегментації на основі VLAN (network-hardware-based)? Яка з них ближча до сучасного розуміння Zero Trust?
4. Чому команда nc від web до db повернула помилку розв'язання імені («bad address»), а не помилку з'єднання? Що це говорить про глибину ізоляції Docker-мереж порівняно із простим блокуванням порту?
5. Поясніть, чому зміна мережевого підключення вже запущеного контейнера (docker network disconnect/connect) не є надійним способом постійної реконфігурації?
6. Що таке mTLS і чим двостороння автентифікація (mutual TLS) принципово відрізняється від звичайного одностороннього TLS (де сертифікат перевіряє лише клієнт)?
7. Чому підключення від «чужого» контейнера з самопідписаним сертифікатом (з тим самим CN, що й легітимний клієнт) було відхилено з помилкою «unknown ca», а не з помилкою невідповідності імені? Що це означає щодо порядку перевірок під час TLS-рукостискання?
8. Наведіть аргумент, чому мережева мікросегментація є необхідною, але недостатньою умовою для Zero Trust, і як Стадія 3 (mTLS) закриває цю прогалину.
9. Чому файл ініціалізації (init-ssl.sh), створений в текстовому редакторі Windows, не виконався в Linux-контейнері з першої спроби? Яка команда допомогла остаточно підтвердити причину?

ЛАБОРАТОРНА РОБОТА 2: Виявлення аномалій мережевого трафіку за допомогою методів машинного навчання

Мета: ознайомитися з двома принципово різними підходами до виявлення шкідливої мережевої активності – сигнатурним (на прикладі алгоритму Random Forest) та поведінковим (на прикладі алгоритму Isolation Forest), і на практиці, на реальному наборі даних, оцінити й порівняти здатність кожного підходу виявляти типи атак, не представлені в навчальній вибірці.

Теоретична частина

Основою сучасної мережевої безпеки є здатність систем виявлення вторгнень (IDS) ефективно відрізняти легітимний трафік від шкідливого. Для автоматизації цього процесу за допомогою машинного навчання традиційно застосовуються два базові підходи: сигнатурний (на основі знань про атаки) та поведінковий (на основі пошуку аномалій). Головна відмінність між підходами полягає в тому, які саме дані використовуються для навчання моделі та як вона реагує на невідомі загрози.

Сигнатурний підхід ґрунтується на розпізнаванні заздалегідь відомих, промаркованих прикладів шкідливої активності. Класифікатор навчається на вибірці, де кожен запис має мітку конкретного типу атаки (або її відсутності), і в подальшому відносить нові записи до одного з відомих класів. Це дозволяє дуже точно ідентифікувати відомі загрози, проте такий класифікатор структурно не може правильно розпізнати те, чого не було в його навчальній вибірці.

Поведінковий (*anomaly-based*) підхід, навпаки, не потребує міток атак узагалі. Модель навчається лише на прикладах нормальної поведінки і позначає аномалією все, що статистично суттєво відхиляється від вивченої норми, незалежно від того, чи бачила модель подібний патерн раніше.

Ключова практична відмінність цих методів проявляється саме на нових, раніше не задокументованих типах атак (0-day атаки): саме поведінковий підхід потенційно здатний визначити будь-яке достатньо сильне відхилення від норми, попередньо не знаючи його конкретної природи, що робить його незамінним для виявлення нових загроз.

Для об’єктивного порівняння цих двох підходів у лабораторній роботі використовується набір даних NSL-KDD, який є вдосконаленою версією класичного набору даних KDD Cup 1999, з якого прибрано надлишкові (дубльовані) записи, що спотворювали оцінку якості моделей у оригінальній версії.

Кожен запис описує одне мережеве з’єднання 41 ознакою, до яких належать тривалість, протокол, службові прапорці, статистичні показники по вікну з’єднань тощо. Також присутнє службове поле *difficulty* (рівень складності класифікації запису за методологією укладачів дата-сету), яке використовується для оцінки, а не як мережева ознака.

Методологічна особливість NSL-KDD, яка й лежить в основі цієї роботи, полягає в тому, що тестова вибірка навмисно містить типи атак, яких немає в тренувальній. Це зроблено спеціально, щоб перевіряти саме здатність моделі до узагальнення на нове, а не просто якість запам’ятовування навчальної вибірки.

Рис. 2.1. Принципова відмінність у навчанні моделей

В роботі порівнюється ефективність двох алгоритмів машинного навчання – Random Forest і Isolation Forest.

Random Forest – це потужний ансамблевий сигнатурний класифікатор, що будує велику кількість дерев рішень на випадкових підвибірках даних і ознак, а підсумкове рішення приймає голосуванням (або усередненням).

Особливості використання для виявлення атак:

— Використання ансамблю дерев суттєво знижує ризик перенавчання порівняно з одним деревом. Алгоритм стійкий до «шуму» в даних та не вимагає складної нормалізації мережевих метрик.

— Дає добру якість на «типових» даних, подібних до навчальної вибірки. Він чудово працює як сигнатурний рушій, безпомилково відносячи відомі патерни (наприклад, стандартні DoS або Probe-атаки) до відповідних класів.

— Дозволяє оцінити важливість кожної ознаки (наприклад, зрозуміти, що кількість невдалих спроб авторизації або певні TCP-прапорці є найважливішими індикаторами конкретної загрози).

Тим не менш, цей алгоритм має певні суттєві недоліки. Наприклад, якщо алгоритм стикається з абсолютно новим типом атаки, він примусово віднесе його до одного з існуючих «листків» дерев (найімовірніше – до легітимного трафіку, якщо атака намагається маскуватися під норму), пропускаючи загрозу.

Isolation Forest є поведінковим аналізатором і вирішує задачу класифікації принципово інакше: замість побудови межі між класами він будує дерева, що випадково розділяють простір ознак. Після цього алгоритм вимірює, скільки розділень (*splits*) знадобилося, щоб «ізолювати» конкретний мережевий запис від решти.

Особливості використання для виявлення атак:

— Аномальні точки (такі, що суттєво відрізняються від основної маси даних) ізолюються значно швидше (за меншу кількість розділень), ніж типові. Легітимний трафік утворює щільні кластери, тому для

відокремлення одного звичайного з'єднання потрібно багато розгалужень. Шкідливий трафік, маючи нетипові атрибути, опиняється «на краях» простору ознак.

— Головна практична перевага для цієї роботи полягає в тому, що Isolation Forest можна навчити, використовуючи лише приклади нормального трафіку, без жодної мітки атаки.

— На відміну від алгоритмів, що вимірюють відстань між точками (наприклад, k-NN), Isolation Forest має дуже низьку обчислювальну складність, що робить його ідеальним для аналізу великих обсягів мережевих логів у реальному часі.

— Модель ефективно виявляє невідомі атаки, оскільки її не цікавить природа аномалії – лише факт того, що з'єднання «не схоже на норму». Проте, це може призводити до хибних спрацьовувань (*false positives*), якщо легітимна поведінка користувачів або топологія мережі раптово змінюється.

В даній роботі будуть використані кілька метрик для проведення оцінки і порівняння досліджуваних алгоритмів (табл. 2.1).

Таблиця 2.1. Метрики якості класифікації

Метрика	**Що показує**	**Чим важлива**
Precision (точність)	Яка частка спрацювань «*attack*» справді є атакою	Низька precision = багато хибних тривог, зайве навантаження на SOC-аналітика
Recall (повнота)	Яка частка реальних атак дійсно виявлена	Головна метрика цієї роботи – саме recall на невідомих атаках показує здатність до узагальнення
ROC-AUC	Наскільки добре модель ранжує атаки вище за норму за ймовірністю, незалежно від порогу	Може залишатися високим навіть тоді, коли конкретний поріг (0.5) погано розпізнає нові типи атак

Практична частина

Робота виконується в середовищі Google Colab (colab.research.google.com).

1. Створення робочого Notebook:

1.1. Відкрити colab.research.google.com → New notebook, перейменувати його для зручності. Виконання відбувається на звичайному Python-рантаймі — прискорювач (GPU) не потрібен, оскільки Random Forest та Isolation Forest на табличних даних такого обсягу рахуються на CPU за секунди (рис. 2.2).

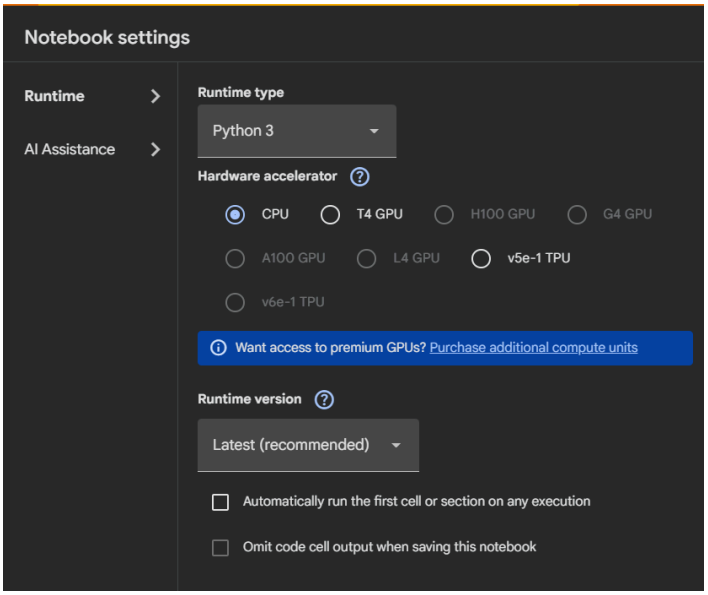


Рис. 2.2. Налаштування Notebook

2. Вибір і завантаження набору даних NSL-KDD:

2.1. На офіційному дзеркалі дата-сету (*Kaggle*)[1] виявилося одразу кілька варіантів файлів. Для базового відтворюваного експерименту потрібні саме повні файли у форматі .txt (KDDTrain+.txt і KDDTest+.txt), які можуть бути зчитані напряму функцією pandas.read_csv без додаткового парсера.

2.2. Завантажуємо файли на ПК, а з ПК – у середовище Google Colab.

```
from google.colab import files
uploaded = files.upload()
```

2.3. Зчитуємо завантажені файли

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, IsolationForest
```

[1] https://www.kaggle.com/datasets/hassan06/nslkdd

```
from sklearn.metrics import classification_report, roc_auc_score,
confusion_matrix, RocCurveDisplay
import matplotlib.pyplot as plt

columns = [

"duration","protocol_type","service","flag","src_bytes","dst_bytes","
land",

"wrong_fragment","urgent","hot","num_failed_logins","logged_in","num_
compromised",

"root_shell","su_attempted","num_root","num_file_creations","num_shel
ls",

"num_access_files","num_outbound_cmds","is_host_login","is_guest_logi
n","count",

"srv_count","serror_rate","srv_serror_rate","rerror_rate","srv_rerror
_rate",

"same_srv_rate","diff_srv_rate","srv_diff_host_rate","dst_host_count"
,

"dst_host_srv_count","dst_host_same_srv_rate","dst_host_diff_srv_rate
",

"dst_host_same_src_port_rate","dst_host_srv_diff_host_rate","dst_host
_serror_rate",

"dst_host_srv_serror_rate","dst_host_rerror_rate","dst_host_srv_rerro
r_rate",
    "label","difficulty"
]

train_df = pd.read_csv("KDDTrain+.txt", names=columns)
test_df = pd.read_csv("KDDTest+.txt", names=columns)

print(train_df.shape, test_df.shape)
```

```
train_df.head()
```

2.4. Проаналізувавши отриманий результат (рис. 2.3), ми бачимо наведені розміри (125973, 43) і (22544, 43) в точності відповідають офіційним повним файлам NSL-KDD: 125973 записи для навчання, 22544 для тестування, по 43 стовпці (41 ознака + label + difficulty). Це означає, що файли завантажені правильно, і дані придатні для подальшої обробки без додаткових перетворень формату.

```
print(train_df.shape, test_df.shape)
train_df.head()
```

```
(125973, 43) (22544, 43)
```

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	0.17	0.03	0.17
1	0	udp	other	SF	146	0	0	0	0	0	...	0.00	0.60	0.88
2	0	tcp	private	S0	0	0	0	0	0	0	...	0.10	0.05	0.00
3	0	tcp	http	SF	232	8153	0	0	0	0	...	1.00	0.00	0.03
4	0	tcp	http	SF	199	420	0	0	0	0	...	1.00	0.00	0.00

5 rows × 43 columns

Рис. 2.3. Результат зчитування файлів KDDTrain+.txt і KDDTest+.txt

3. Аналіз розподілу міток і пошук невідомих типів атак:

```
train_labels = set(train_df["label"].unique())
test_labels = set(test_df["label"].unique())
unseen_attacks = test_labels - train_labels
print(len(unseen_attacks))
print(unseen_attacks)
```

Провівши аналіз тестового і навчального наборів, було показано, що в тестовому наборі присутні 17 типів атак, які відсутні в тренувальному, а саме: xterm, sqlattack, ps, processtable, xlock, snmpgetattack, xsnoop, snmpguess, apache2, mscan, saint, sendmail, worm, mailbomb, named, udpstorm, httptunnel. Це означає, що дата-сет NSL-KDD навмисно спроєктований таким чином, щоб перевіряти узагальнювальну здатність моделі, а не просто якість запам’ятовування. Ці 17 типів стануть ключовою тестовою підмножиною для порівняння підходів далі в роботі, саме на них очікується найбільша розбіжність між сигнатурним і поведінковим підходами.

4. Кодування категоріальних ознак:

4.1. Оскільки бібліотека scikit-learn, яка використовується для реалізації алгоритмів Random Forest і Isolation Forest, працює виключно з числовими даними, постає необхідність кодування категоріальних ознак (в нашому випадку мова йде про категорії protocol_type, service, flag) (рис. 2.4). Текстові значення необхідно перетворити на числові (наприклад, на цілі числа 0, 1, 2... за допомогою методу Label Encoding або на бінарні вектори за допомогою One-Hot Encoding).

```
from sklearn.preprocessing import LabelEncoder
categorical_cols = ["protocol_type", "service", "flag"]
combined = pd.concat([train_df[categorical_cols],
test_df[categorical_cols]], axis=0)
for col in categorical_cols:
    le = LabelEncoder()
    le.fit(combined[col])
    train_df[col] = le.transform(train_df[col])
    test_df[col] = le.transform(test_df[col])
```

Зауважимо, що кодувальник навмисно навчався на об’єднаних train + test значеннях категорій (а не лише на тренувальних) – це запобігає помилці «*unseen label*», якщо в тестовій вибірці трапиться значення служби, відсутнє в тренувальній.

Кодування завершено

	protocol_type	service	flag
0	1	20	9
1	2	44	9
2	1	49	5
3	1	24	9
4	1	24	9

Рис. 2.4. Результат кодування категоріальних ознак

5. Бінаризація цільових міток для алгоритму Random Forest:

5.1. З метою спрощення першої моделі до бінарної задачі: «нормальний трафік» проти «будь-яка атака» (замість десятків окремих класів атак) проведемо бінарізацію цільових міток (рис. 2.5).

```
train_df["is_attack"] = (train_df["label"] !=
"normal").astype(int)
test_df["is_attack"] = (test_df["label"] !=
"normal").astype(int)
print(train_df["is_attack"].value_counts())
print(test_df["is_attack"].value_counts())
```

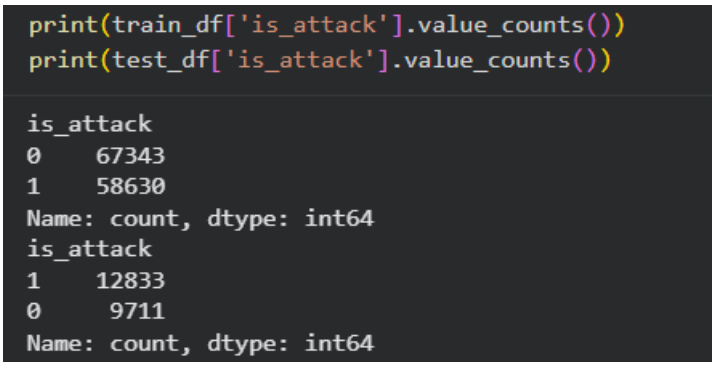


Рис. 2.5. Результат бінарізації цільових міток

Аналіз отриманих даних показав, що обидва класи (normal/attack) представлені порівнянними частками – суттєвого дисбалансу класів немає, тому надалі можна орієнтуватися на просту *accuracy* як змістовну (хоча й не єдину) метрику, без ризику, що вона буде оманливо високою через переважання одного класу.

6. Навчання Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, roc_auc_score

feature_cols = [c for c in train_df.columns if c not in
["label", "difficulty", "is_attack"]]
X_train, y_train = train_df[feature_cols], train_df["is_attack"]
X_test, y_test = test_df[feature_cols], test_df["is_attack"]
```

```
rf = RandomForestClassifier(n_estimators=100, random_state=42,
n_jobs=-1)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print(classification_report(y_test, y_pred_rf,
target_names=["normal","attack"]))
print("ROC-AUC:", roc_auc_score(y_test,
rf.predict_proba(X_test)[:,1]))
```

```
••• === Random Forest: результати на тестовій вибірці ===
              precision    recall  f1-score   support

      normal       0.66      0.97      0.78      9711
      attack       0.97      0.62      0.75     12833

    accuracy                           0.77     22544
   macro avg       0.81      0.80      0.77     22544
weighted avg       0.83      0.77      0.77     22544

ROC-AUC: 0.9620187888803534
```

Рис. 2.6. Отримані результати навчання Random Forest

Аналіз результатів: на перший погляд результат хороший – ROC-AUC на рівні 0.96 свідчить, що модель загалом добре ранжує атаки вище за норму за ймовірністю, але повнота (*recall*) для attack (0.62) уже помітно нижча за повноту для normal (0.97), тобто модель має явний перекос у бік пропуску атак, а не хибних спрацювань.

Поле *difficulty* свідомо виключене зі списку ознак (feature_cols), оскільки це службовий параметр самого дата-сету, а не характеристика мережевого з'єднання, і її використання для навчання було б методологічною помилкою.

7. Перевірка Random Forest на невідомих типах атак:

```
unseen_mask = test_df["label"].isin(unseen_attacks)
y_test_unseen = y_test[unseen_mask]
y_pred_unseen = y_pred_rf[unseen_mask]
print(classification_report(y_test_unseen, y_pred_unseen,
target_names=["normal","attack"], zero_division=0))
```

```
Записів з невідомими типами атак у тесті: 3750

=== Random Forest: ЛИШЕ на невідомих типах атак ===
              precision    recall  f1-score   support

      normal       0.00      0.00      0.00         0
      attack       1.00      0.27      0.43      3750

    accuracy                           0.27      3750
   macro avg       0.50      0.14      0.21      3750
weighted avg       1.00      0.27      0.43      3750
```

Рис. 2.7. Результати Random Forest на невідомих типах атак

Аналіз результатів: на невідомих типах атак якість моделі впала майже вдвічі порівняно із загальним показником повноти 0.62 – Random Forest пропустив 73% реальних атак, класифікувавши їх як normal, оскільки нічого схожого не бачив під час навчання. Високий ROC-AUC (0.96) під час навчання і низький показник повноти (0.27) на невідомих атаках не суперечать один одному: перший показник вимірює загальну якість ранжування на всій вибірці (де більшість атак усе ж знайомі моделі), другий – конкретно здатність до узагальнення на справді нове. Значення precision=1.00 тут – математичний артефакт відсутності класу normal у цій підмножині (усі 3750 записів – реальні атаки), а не ознака якості моделі, і не повинно інтерпретуватися як «модель ідеально працює на невідомих атаках».

8. Навчання Isolation Forest

```
from sklearn.ensemble import IsolationForest

iso = IsolationForest(n_estimators=100, contamination=0.1,
random_state=42, n_jobs=-1)
iso.fit(X_train[y_train == 0])  # лише нормальний трафік, без
міток атак
y_pred_iso = (iso.predict(X_test) == -1).astype(int)
```

```
print(classification_report(y_test, y_pred_iso,
target_names=["normal","attack"]))

y_pred_iso_unseen = y_pred_iso[unseen_mask]
print(classification_report(y_test_unseen, y_pred_iso_unseen,
target_names=["normal","attack"], zero_division=0))
```

```
=== Isolation Forest: результати на ВСІЙ тестовій вибірці ===
              precision    recall  f1-score   support

      normal       0.72      0.92      0.81      9711
      attack       0.93      0.73      0.82     12833

    accuracy                           0.81     22544
   macro avg       0.83      0.83      0.81     22544
weighted avg       0.84      0.81      0.82     22544


=== Isolation Forest: ЛИШЕ на невідомих типах атак ===
              precision    recall  f1-score   support

      normal       0.00      0.00      0.00         0
      attack       1.00      0.77      0.87      3750

    accuracy                           0.77      3750
   macro avg       0.50      0.38      0.43      3750
weighted avg       1.00      0.77      0.87      3750
```

Рис. 2.8. Результати відпрацювання Isolation Forest

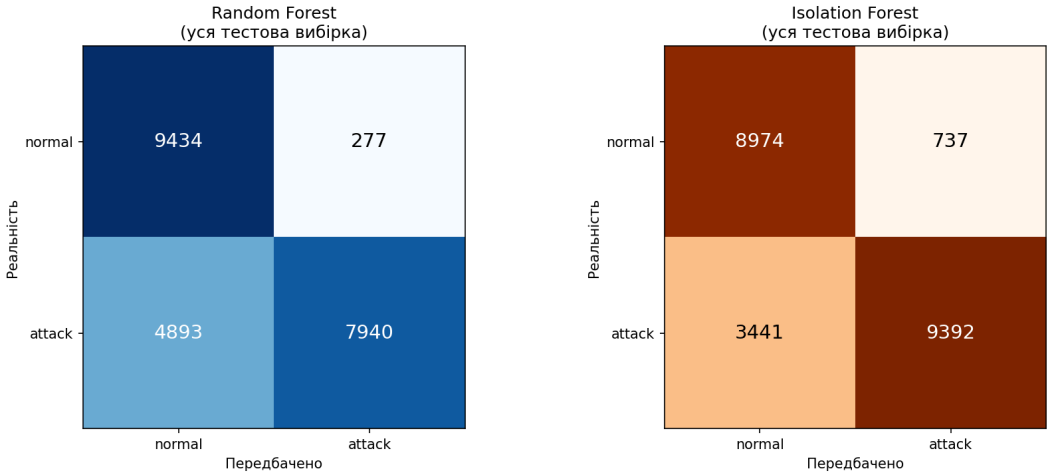
Аналіз результатів: на тих самих 17 невідомих типах атак Isolation Forest досяг значення повноти 0.77 проти 0.27 у Random Forest, що є майже потрійною перевагою. Модель, яка під час навчання взагалі не отримувала жодної мітки атаки (лише приклади normal), впоралася з абсолютно новими типами загроз значно краще за класифікатор, навчений із мітками, – оскільки вона вивчила загальний принцип «як виглядає норма», а не запам’ятала конкретні патерни відомих атак.

Зауважимо наступне, Isolation Forest дав recall=0.92 для normal, але це означає 8% хибних спрацювань на цілком легітимному трафіку (false positives), тоді як Random Forest на normal мав recall=0.97 (тобто менше хибних спрацювань) – це є ілюстрацією класичного компромісу security-аналітики: anomaly-based підходи ловлять більше невідомого, ціною

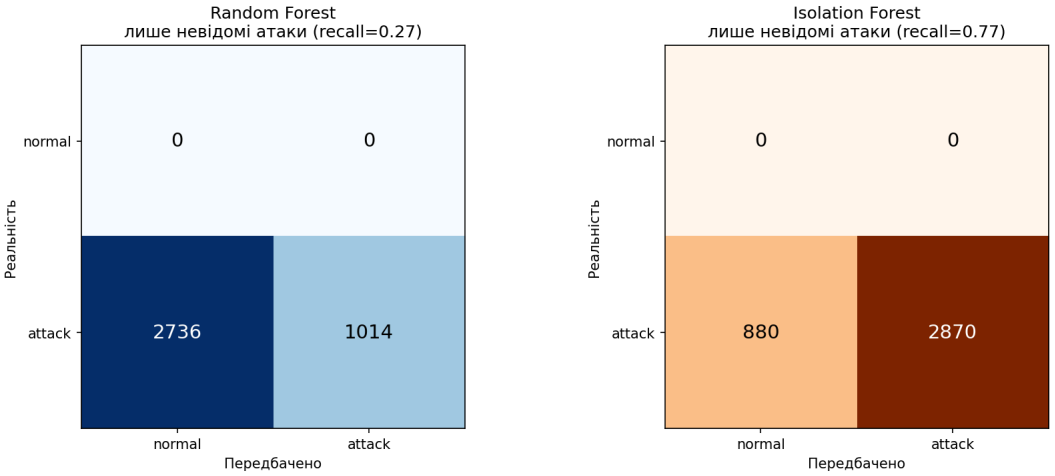
більшої кількості «шуму», який доведеться розбирати SOC-аналітику вручну. Обидва підходи мають реальну практичну цінність – саме тому в індустрії часто використовують їх разом (сигнатурний для швидкого, точного відсіювання відомого; anomaly-based як другий шар для незнайомого), а не замінюють один одним.

9. Побудова матриць помилок (Confusion Matrix)

```
from sklearn.metrics import confusion_matrix
print("RF (вся вибірка):\n", confusion_matrix(y_test,
y_pred_rf))
print("ISO (вся вибірка):\n", confusion_matrix(y_test,
y_pred_iso))
print("RF (unseen):\n", confusion_matrix(y_test_unseen,
y_pred_unseen))
print("ISO (unseen):\n", confusion_matrix(y_test_unseen,
y_pred_iso_unseen))
```



a



b

Рис. 2.9. Матриці помилок для випадку класифікації тестового набору (a) і для випадку класифікації лише невідомих атак (b)

Аналіз результатів: числа наочно показують компроміс двох підходів. На всій вибірці Random Forest дає значно менше хибних тривог (277 проти 737), тобто менше «шуму» для SOC-аналітика – але пропускає більше реальних атак (4893 проти 3441). Isolation Forest – дзеркально навпаки: більше хибних тривог, зате суттєво менше пропущених атак, і ця перевага стає вирішальною саме на підмножині невідомих атак (880 пропущених проти 2736). Жодна з моделей не є «краще» в абсолютному сенсі – вибір залежить від того, що дорожче в конкретному середовищі: пропущена атака чи хибна тривога.

10. Побудова ROC-кривої Random Forest:

```
from sklearn.metrics import RocCurveDisplay
import matplotlib.pyplot as plt
RocCurveDisplay.from_predictions(y_test,
rf.predict_proba(X_test)[:,1])
plt.plot([0,1],[0,1], linestyle="--", color="gray")
plt.savefig("roc_curve_rf.png", dpi=150, bbox_inches="tight")
plt.show()
```

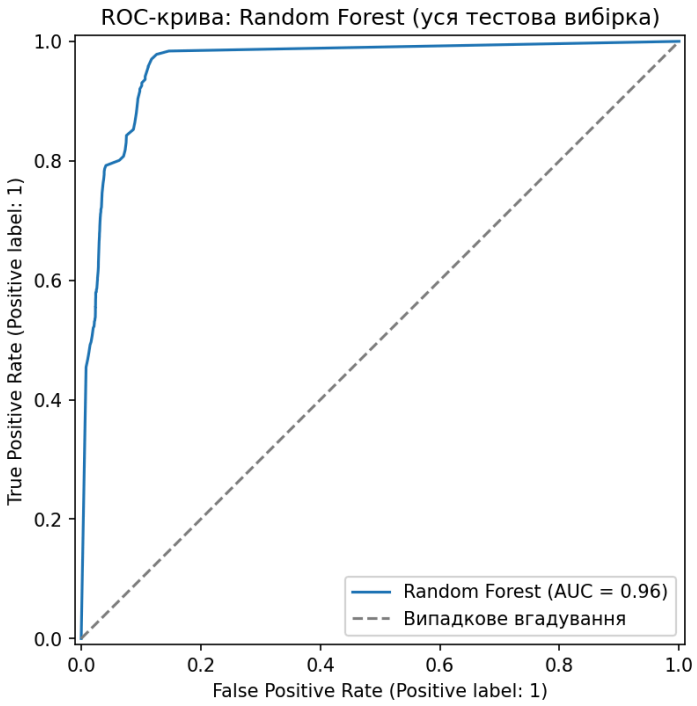


Рис. 2.10. ROC-крива Random Forest

Аналіз результатів: крива швидко піднімається до True Positive Rate ≈ 0.95+ вже за малого False Positive Rate, AUC = 0.96, тобто Random Forest добре ранжує атаки за ймовірністю в цілому по вибірці. Слабкість Random Forest проявляється не в загальній розрізняльній здатності, а конкретно в узагальненні на типи атак поза навчальною вибіркою.

Методологічне зауваження: аналогічну ROC-криву для Isolation Forest свідомо не будувалась для прямого порівняння, оскільки decision_function цієї моделі повертає міру «аномальності» у довільному, некаліброваному масштабі, тоді як predict_proba класифікатора – це справжня ймовірність класу. Пряме зіставлення двох ROC-кривих на одному графіку без додаткової калібровки було б методологічно некоректним; для порівняння моделей використано classification_report на однакових вибірках.

Індивідуальні завдання

Кожен здобувач отримує індивідуальний варіант (табл. 2.2) — власну фокус-категорію атак за стандартною 4-частинною класифікацією NSL-KDD

(DoS, Probe, R2L, U2R; наведений розподіл міток є орієнтовним і уточнюється за документацією дата-сету), альтернативний класифікатор для порівняння з Isolation Forest, і власне значення параметра забруднення даних (*contamination*).

Завдання:

— виділити зі своєї категорії ті типи атак, що потрапляють до множини unseen_attacks;

— навчити свій альтернативний класифікатор на тих самих ознаках і порівняти його classification_report (загалом і окремо на невідомих атаках свого варіанта) з уже отриманими результатами Random Forest та Isolation Forest;

— навчити Isolation Forest із призначеним значенням contamination і пояснити в звіті, як зміна цього параметра вплинула на баланс precision/recall порівняно з базовим значенням 0.1 з основної частини роботи;

— оформити звіт із таблицями метрик і короткими висновками щодо своєї категорії атак.

Таблиця 2.2. Варіанти індивідуальних завдань

№	**Фокус-категорія атак**	**Альтернативний класифікатор для порівняння з Isolation Forest**	**Contamination**
1	DoS (back, neptune, smurf, teardrop, pod, apache2, udpstorm, processtable, worm, mailbomb, land)	GradientBoostingClassifier	0.10
2	Probe (satan, ipsweep, nmap, portsweep, mscan, saint)	LogisticRegression	0.15
3	R2L (guess_passwd, ftp_write, imap, phf, multihop, warezmaster, warezclient, spy, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named)	SVC (Support Vector Classifier)	0.05
4	U2R (buffer_overflow, loadmodule, perl, rootkit, ps, sqlattack, xterm)	KNeighborsClassifier	0.20
5	DoS + Probe (об’єднана категорія)	один DecisionTreeClassifier (без ансамблю)	0.10

6	R2L + U2R (найрідкісніші й найскладніші класи)	RandomForestClassifier(class_weight='balanced')	0.05

Контрольні питання

1. Чому саме recall на підмножині невідомих типів атак є головною метрикою якості в цій роботі, а не загальна accuracy?
2. Поясніть, чому висока ROC-AUC (0.96) Random Forest не суперечить низькому recall (0.27) на невідомих атаках? Які різні властивості моделі вимірюють ці дві метрики?
3. Чому Isolation Forest навчається лише на прикладах класу normal, тоді як Random Forest потребує прикладів обох класів? Як це впливає на здатність кожної моделі виявляти нові, раніше не бачені типи атак?
4. Чому LabelEncoder для категоріальних ознак (protocol_type, service, flag) навчався на об’єднаних train + test даних?
5. Чому колонку difficulty було виключено зі списку ознак для навчання моделей?
6. Поясніть компроміс precision/recall між Random Forest та Isolation Forest на повній тестовій вибірці: яка модель дає менше хибних тривог, а яка — менше пропущених атак?
7. Чому precision класу attack дорівнював рівно 1.00 при перевірці лише на невідомих атаках, і чому це не варто сприймати як ознаку ідеальної роботи моделі?
8. Що таке параметр contamination в Isolation Forest і як його зміна вплине на баланс між кількістю хибних тривог і пропущених атак?
9. Чому в цій роботі не проводилося пряме порівняння ROC-кривих Random Forest та Isolation Forest на одному графіку?
10. Яка практична порада випливає з результатів цієї роботи щодо застосування сигнатурних та anomaly-based методів у реальному SOC? Чи варто обирати один із двох підходів, чи використовувати обидва?

ЛАБОРАТОРНА РОБОТА 3: Активний пошук загроз за матрицею MITRE ATT&CK

Мета: ознайомитися з методами активного пошуку загроз на основі готових Sigma-правил, промаркованих техніками MITRE ATT&CK, використовувати інструмент Hayabusa для аналізу реальних історичних журналів Windows (EVTX), і навчитися складати власні правила виявлення для активності, яку жодне готове правило не покриває.

Теоретична частина

Сучасна кібербезпека вимагає переходу від виключно реактивного захисту до активного виявлення загроз. Інструменти автоматизованого контролю (антивіруси, базові правила SIEM/IDS) ловлять більшість типових атак, проте складні та цілеспрямовані загрози (APT) можуть обходити стандартні сигнатури. Для їх виявлення застосовується методологія активного пошуку загроз.

Активний пошук загроз (*threat hunting*) – це проактивний, гіпотезо-орієнтований підхід до виявлення шкідливої активності, на відміну від реактивного реагування на вже сформовані алерти SIEM/IDS (рис. 3.1).

Процес будується навколо аналітика, який формулює гіпотезу («*якщо в мережі присутній зловмисник, які сліди він мав би залишити*») і цілеспрямовано шукає підтвердження чи спростування в наявних журналах. Це робиться незалежно від того, чи спрацювало для цього якесь готове правило.

Для того, щоб гіпотези були структурованими, аналітики спираються на базу знань MITRE ATT&CK – глобальну матрицю тактик (цілей зловмисника) та технік (конкретних способів досягнення цих цілей) і слугує спільною мовою кібербезпеки, дозволяючи класифікувати будь-яку виявлену аномалію.

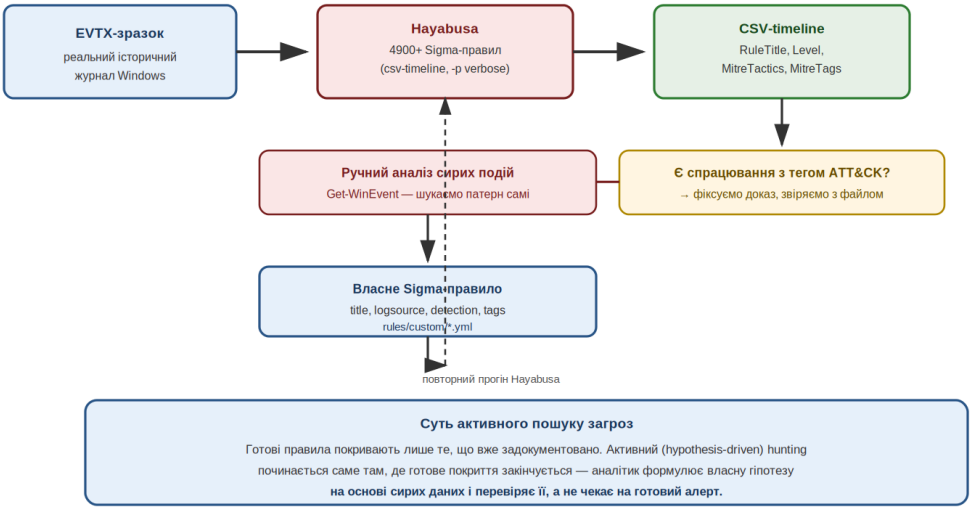


Рис. 3.1. Цикл активного пошуку загроз

Для автоматизації пошуку слідів компрометації та обміну експертизою в індустрії використовується Sigma – вендоронезалежний, відкритий формат опису правил виявлення (YAML), що дозволяє один раз описати логіку детекції й транслювати її під конкретний SIEM або інструмент аналізу.

Кожне правило складається з кількох логічних блоків:

— Секція логіки (*detection*): безпосередньо описує патерни подій, які потрібно знайти (наприклад, запуск певного процесу з підозрілими аргументами).

— Метадані: рівень критичності (*level*) та опис (*description*), які допомагають аналітику зрозуміти суть загрози.

— Теги: ключовий елемент, що прив'язує правило до конкретної тактики й техніки MITRE ATT&CK (наприклад, attack.t1003 – OS Credential Dumping).

Для практичного застосування Sigma-правил на локальних артефактах (без розгортання важких SIEM-систем) використовується програмний комплекс Hayabusa (Yamato Security). Це інструмент з відкритим кодом для швидкого аналізу журналів подій Windows (EVTX) великим набором Sigma-правил. У стандартній поставці програми міститься понад 4900 таких правил. Головна перевага інструменту полягає в тому, що він зводить сотні

тисяч сирих подій у зручний формат: Hayabusa виводить результат у вигляді єдиного хронологічного timeline із зазначенням спрацьованого правила, його критичності та відповідних тегів ATT&CK.

Щоб навчитися працювати з цими інструментами, необхідні якісні дані, що містять реальні сліди атак. У цій лабораторній роботі використовується EVTX-ATTACK-SAMPLES (автор – sbousseaden) – публічний, стабільний репозиторій реальних, історично захоплених журналів Windows. Кожен файл у цьому наборі відповідає конкретній, задокументованій техніці атаки. Використання саме цього репозиторію має суттєву перевагу над безкоштовними онлайн-SOC-платформами для дистанційного навчання. Оскільки доступність останніх повністю залежить від комерційної політики стороннього сервісу, а цей репозиторій (як відкритий код на GitHub) можна клонувати собі й використовувати необмежено довго, незалежно від змін у чужих безкоштовних тарифах.

У рамках цієї лабораторної роботи буде пройдено повний цикл: від автоматизованого аналізу історичних журналів за допомогою Hayabusa до написання власних кастомних Sigma-правил для загроз, які вислизають від стандартного виявлення.

Практична частина

1. Встановлення і налаштування програмного комплексу Hayabusa:

1.1. Завантажити останній реліз з GitHub за посиланням https://github.com/Yamato-Security/hayabusa/releases. Розпакувати у довільну теку.

1.2. Оновити набір Sigma-правил

```
.\hayabusa-3.10.0-win-x64.exe update-rules
```

і перевірити фактичну кількість завантажених правил

```
(Get-ChildItem -Path .\rules -Filter *.yml -Recurse).Count
```

Очікуваний результат – близько 4900+ файлів правил.

1.3. Завантажити зразки EVTX-ATTACK-SAMPLES.

Важливо! Пряме завантаження архіву репозиторію через кнопку «Download ZIP» у браузері може бути заблоковане попередженням «Virus detected», оскільки репозиторій містить десятки реальних лог-файлів із командними рядками справжніх red-team-інструментів (mimikatz, PsExec, Cobalt Strike-подібна активність). Це призводить до того, що евристичний сканер браузера (наприклад, Google Safe Browsing) іноді реагує на сукупний обсяг такого вмісту в одному великому архіві. Найнадійніший обхідний шлях – клонування через Git, яке не проходить через той самий механізм перевірки:

```
git clone https://github.com/sbousseaden/EVTX-ATTACK-SAMPLES.git
C:\LABs\4\samples
```

1.4. Оглянути структуру зразків.

```
Get-ChildItem C:\LABs\4\samples -Directory
```

Тека вже структурована за категоріями, що майже дослівно відповідають тактикам MITRE ATT&CK (рис. 3.2): Credential Access, Defense Evasion, Discovery, Execution, Lateral Movement, Persistence, Privilege Escalation, Command and Control, AutomatedTestingTools – це й формує основу для індивідуальних завдань цієї роботи.

```
PS C:\LABs\4\hayabusa> Get-ChildItem C:\LABs\4\samples -Directory


    Directory: C:\LABs\4\samples


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        25.07.2026      0:52                .vscode
d-----        25.07.2026      0:52                AutomatedTestingTools
d-----        25.07.2026      0:52                Command and Control
d-----        25.07.2026      0:52                Credential Access
d-----        25.07.2026      0:52                Defense Evasion
d-----        25.07.2026      0:52                Discovery
d-----        25.07.2026      0:52                EVTX_ATT&CK_Metadata
d-----        25.07.2026      0:52                Execution
d-----        25.07.2026      0:52                Lateral Movement
d-----        25.07.2026      0:52                Other
d-----        25.07.2026      0:52                Persistence
d-----        25.07.2026      0:52                Privilege Escalation
```

Рис. 3.2. Структура теки із зразками

2. Перше полювання (файл із категорії Credential Access):

2.1. Обрати файл CA_Mimikatz_Memssp_Default_Logs_Sysmon_11.evtx, пов’язаний з технікою mimikatz::misc::memssp (T1556.002) і виконати перший аналіз.

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -f
"C:\LABs\4\samples\Credential
Access\CA_Mimikatz_Memssp_Default_Logs_Sysmon_11.evtx" -o
results_mimikatz.csv -U
```

Зауваження: при першому запуску з’являється інтерактивний «Scan wizard» із питаннями про набір правил для завантаження. Рекомендовано обрати варіант «4. All alert rules» – найширше практичне покриття без зайвих суто інформаційних подій. На всі інші питання візарда слід обирати запропоновані відповіді.

Рис. 3.3. Інтерактивний «Scan wizard»

2.2. Провести аналіз результатів першого полювання

```
Import-Csv results_mimikatz.csv | Select-Object Timestamp,
RuleTitle, Level, MitreTactics, MitreTags | Format-List
```

Отриманий результат демонструє одне спрацювання – «**HackTool – Mimikatz Kirbi File Creation**», рівень – critical (рис. 3.4). Однак поля MitreTactics і MitreTags виявилися порожніми.

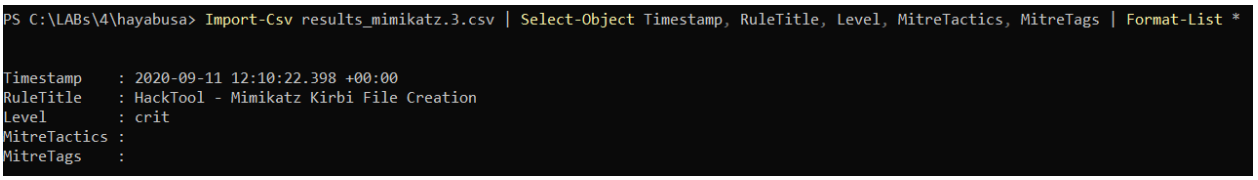


Рис. 3.4. Спрацювання на загрозу «HackTool – Mimikatz Kirbi File Creation»

Зауваження: перевірка через Format-List * (виведення геть усіх полів запису) підтвердила, що це не обрізаний вивід терміналу, а справжня відсутність заповнених тегів для цього конкретного правила при використанні профілю виводу за замовчуванням.

3. Друге полювання (файл із категорії Discovery – BloodHound):

3.1. Обрати файл discovery_bloodhound.evtx, пов'язаний з тактикою Discovery (TA0007) і виконати аналіз.

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -f
"C:\LABs\4\samples\Discovery\discovery_bloodhound.evtx" -o
results_bloodhound.csv -U
```

3.2. Провести аналіз результатів другого полювання

```
Import-Csv results_bloodhound.csv | Select-Object Timestamp,
RuleTitle, Level, MitreTactics, MitreTags | Format-List
```

Отриманий результат демонструє одне спрацювання – «**Log Cleared**» (Event ID 1102, очищення журналу безпеки, що є ознакою замітання слідів), знову без заповнених тегів.

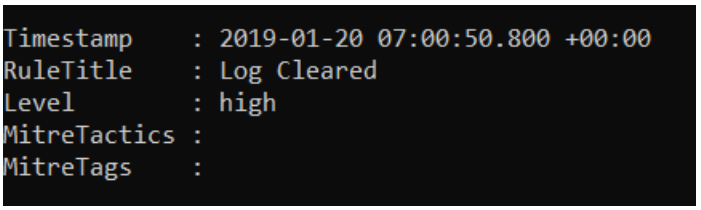


Рис. 3.5. Спрацювання на загрозу «Log Cleared»

4. Третє полювання (файл із категорії Lateral Movement – PsExec/Meterpreter):

4.1. Обрати файл LM_sysmon_psexec_smb_meterpreter.evtx і виконати аналіз

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -f
"C:\LABs\4\samples\Lateral
```

```
Movement\LM_sysmon_psexec_smb_meterpreter.evtx" -o
results_psexec.csv -U
```

Важливо: на цьому кроці Windows Defender заблокував (перевів у карантин) уже не вхідний EVTX-файл, а сам щойно створений результуючий CSV – команда Import-Csv видала помилку «The file contains a virus», а сам файл results_psexec.csv фізично зник з диска. Причина – Hayabusa перенесла в колонку Details похідного CSV сирі командні рядки з події Sysmon, що відповідають артефактам Meterpreter, і саме ця похідна текстова комбінація викликала спрацювання антивіруса, а не вихідний EVTX-файл. Рішенням цієї проблеми є створення тимчасового винятку для робочої теки, знятий одразу після завершення роботи:

```
Add-MpPreference -ExclusionPath "C:\LABs\4"
```

Повторний запуск після встановлення винятку пройшов успішно.

Як і під час двох попередніх полювань отримані результати не демонструють теги MitreTactics і MitreTags (рис. 3.6).

```
Timestamp    : 2019-04-30 20:26:52.356 +00:00
RuleTitle    : Suspicious PowerShell Invocations - Specific - ProcessCreation
Level        : med
MitreTactics :
MitreTags    :

Timestamp    : 2019-04-30 20:26:52.106 +00:00
RuleTitle    : Non Interactive PowerShell Process Spawned
Level        : low
MitreTactics :
MitreTags    :

Timestamp    : 2019-04-30 20:26:52.356 +00:00
RuleTitle    : Non Interactive PowerShell Process Spawned
Level        : low
MitreTactics :
MitreTags    :
```

Рис. 3.6. Частковий результатів аналізу файлу LM_sysmon_psexec_smb_meterpreter.evtx

5. Визначення причини порожніх тегів.

Вивід CSV у Hayabusa залежить від обраного профілю (`--profile`): типовий профіль не включає повноцінно заповнені колонки ATT&CK, тоді як

профіль `verbose` – включає. Перевірка проведена повторно на файлі PsExec з явним вказанням профілю:

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -f
"C:\LABs\4\samples\Lateral
Movement\LM_sysmon_psexec_smb_meterpreter.evtx" -o
results_psexec_verbose.csv -p verbose -U --no-wizard
Import-Csv results_psexec_verbose.csv | Select-Object Timestamp,
RuleTitle, Level, MitreTactics, MitreTags | Format-List
```

Отриманий результат цього разу демонструє понад 30 спрацювань з повністю заповненими полями, зокрема «Potential CobaltStrike Service Installations – Registry» з тегами MitreTactics: Persis | Exec | PrivEsc | LatMov та MitreTags: T1021.002 | T1543.003 | T1569.002 (рис. 3.7).

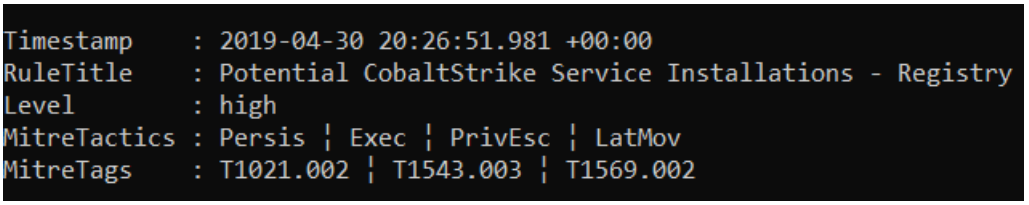
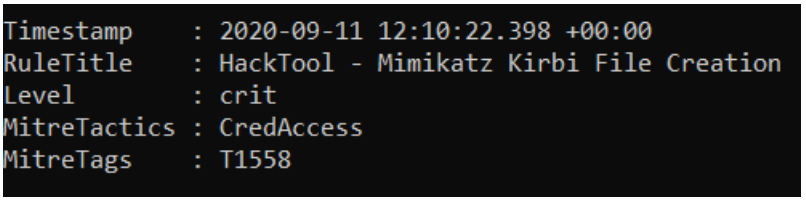
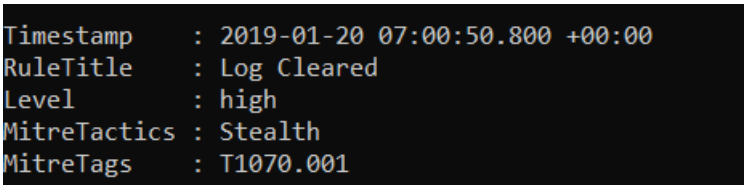


Рис. 3.7. Аналіз файлу LM_sysmon_psexec_smb_meterpreter.evtx з прапорцем `-p verbose`

Контрольна перевірка: повторний прогін файлів із п. 2.2 і 3.2 із прапорцем `-p verbose` підтвердив визначену причину остаточно – «Mimikatz Kirbi» дав тег T1558, а «Log Cleared» – T1070.001.



a



b

Рис. 3.8. Результат повторного аналізу файлів з п. 2.2 (a) і 3.2 (b) при використанні прапорця `-p verbose`

Відтепер прапорець `-p verbose` – обов'язковий параметр для всіх подальших прогонів.

6. Масове полювання по цілій категорії:

Hayabusa дозволяє проаналізувати цілу теку одним викликом (прапорець `-d` замість `-f`), що є значно ефективнішим за послідовний аналіз файлів по одному.

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -d
"C:\LABs\4\samples\Lateral Movement" -o results_lateral_all.csv
-p verbose -U --no-wizard
Import-Csv results_lateral_all.csv | Group-Object RuleTitle |
Sort-Object Count -Descending | Select-Object Count, Name |
Format-Table -AutoSize
```

Отримання результату аналізу близько 40 файлів зайняло ~3.1 секунди; знайдено 1 критичний алерт (Malicious Named Pipe Created), найчастіший high-алерт – RunMRU Registry Key Deletion (21 випадок) і HackTool – Potential Impacket Lateral Movement Activity (9 випадків); у розподілі уражених хостів видно кілька різних імен (IEWIN7, MSEDGEWIN10, DC1.insecurebank.local тощо), що підтверджує: тека зібрана з незалежних тестових захоплень різних дослідників DFIR-спільноти.

7. Написання власного правила:

Іноді виникають ситуацію, коли готового покриття правил виявляється недостатньо. Так, незважаючи на спрацювання «Log Cleared» у файлі

discovery_bloodhound.evtx (п. 3.2), сама розвідувальна активність BloodHound не викликала жодного окремого правила.

7.1. Перш ніж писати правило навмання, досліджено сирі події файлу:

```
Get-WinEvent -Path
"C:\LABs\4\samples\Discovery\discovery_bloodhound.evtx" -Oldest
| Select-Object -First 20 Id, Message | Format-List
```

Виявлено серію подій Event ID 5145 (перевірка доступу до мережевого спільного ресурсу) від одного джерела (10.0.2.17), що за секунди послідовно звертається через IPC$[2] до іменованих каналів samr → lsarpc → NETLOGON → samr → samr → srvsvc, що є характерною мережевою сигнатурою інструментів розвідки Active Directory (на кшталт, BloodHound/SharpHound), незалежною від назви процесу (в журналі Security імен процесів немає в принципі – лише в Sysmon).

7.2. Робоча версія правила:

```
author: <ім'я автора>
date: 2026-07-25
title: Potential AD Enumeration via SAMR/LSARPC Named Pipe
Access
id: 8f3e6a1c-9d2b-4e77-b1a4-6c8e2f0d5a91
status: experimental
description: 'Detects AD enumeration via SAMR/LSARPC named pipe
access over IPC$'
logsource:
  product: windows
  service: security
detection:
    selection:
        Channel: Security
        EventID: 5145
        ShareName|contains: 'IPC$'
        RelativeTargetName:
            - 'samr'
            - 'lsarpc'
            - 'netlogon'
    condition: selection
level: medium
```

[2] IPC$ (Inter-Process Communication) — це спеціальний прихований адміністративний ресурс (шар) в операційній системі Windows. Він не має реальної фізичної папки на диску, а використовується для забезпечення зв'язку між різними програмами та процесами на різних комп'ютерах у мережі за допомогою протоколу SMB та іменованих каналів (named pipes).

```
tags:
  - attack.discovery
  - attack.t1087.002
```

7.3. Перевірка роботи створеного правила:

```
.\hayabusa-3.10.0-win-x64.exe csv-timeline -f
"C:\LABs\4\samples\Discovery\discovery_bloodhound.evtx" -o
results_final.csv -p verbose -U --no-wizard -C
Import-Csv results_final.csv | Select-Object RuleTitle,
MitreTactics, MitreTags | Format-List
```

Отриманий результат демонструє, що створене правило «Potential AD Enumeration via SAMR/LSARPC Named Pipe Access» (MitreTactics: Disc, MitreTags: T1087.002) спрацювало 7 разів, підтверджуючи серію знайдених вручну подій, поруч із незмінним «Log Cleared» (T1070.001). Таким чином файл, що раніше видавався «непокритим» готовими правилами щодо основної (розвідувальної) активності, тепер має повне, підтверджене покриття обох компонентів інциденту: розвідки та замітання слідів.

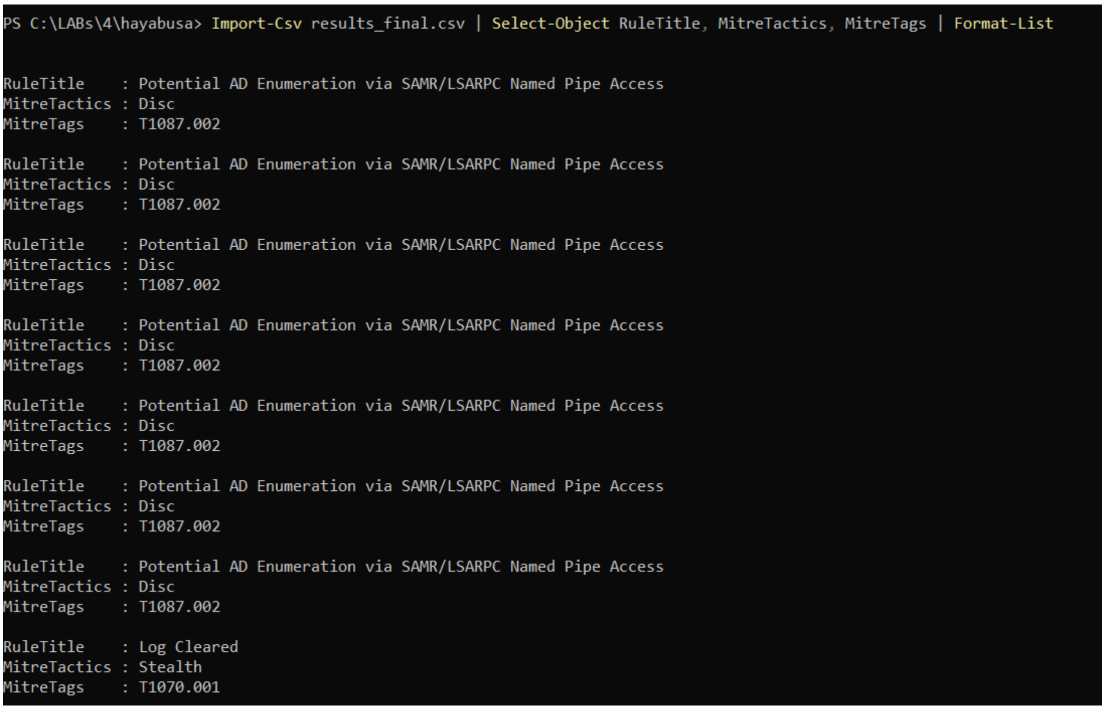


Рис. 3.9. Результат спрацювання власного правила

Індивідуальні завдання

Кожен здобувач отримує окрему тематичну теку репозиторію EVTX-ATTACK-SAMPLES (табл. 3.1). Завдання виконується за тим самим циклом, що й основна частина роботи:

— обрати кілька файлів свого варіанта й виконати аналіз (`-f` для окремого файлу, `-d` для всієї теки), обов'язково з прапорцем `-p verbose`;

— зафіксувати знайдені спрацювання з тегами MitreTactics/MitreTags;

— визначити, чи є в обраних файлах активність, що НЕ отримала жодного спрацювання від готових правил (як це сталося з BloodHound в основній частині);

— якщо так – дослідити сирі події (Get-WinEvent), скласти власне Sigma-правило і підтвердити його роботу;

— оформити звіт зі скріншотами кожного етапу.

Таблиця 3.1. Варіанти індивідуальних завдань

№	Тека EVTX-ATTACK-SAMPLES	Фокус завдання
1	Credential Access	Обрати 2-3 файли, пов’язані з різними інструментами дампу облікових даних (mimikatz, keeper тощо); порівняти, чи всі спрацьовані правила мають заповнені MitreTags.
2	Defense Evasion	Знайти файл, пов’язаний із очищенням/маніпуляцією журналами або обфускацією (наприклад, base64-команди); зіставити з T1070/T1027.
3	Discovery	Обрати файл, відмінний від discovery_bloodhound.evtx (уже розібраний у прикладі); перевірити, чи готові правила покривають знайдену активність повністю.
4	Execution	Знайти приклад виконання через нестандартний інтерпретатор (PowerShell, WMI, scheduled task) і зіставити з відповідною технікою T1059.x.
5	Lateral Movement — фокус PsExec/WMI	Обрати 2-3 файли з назвами, що містять psexec/wmi/impacket; порівняти покриття різних інструментів для того самого T1021.
6	Lateral Movement — фокус RDP/WinRM	Обрати файли, пов'язані з RDP (tsclient) чи WinRM/PowerShell Remoting; зіставити з T1021.001/T1021.006.
7	Persistence	Знайти приклад закріплення через реєстр, заплановане завдання чи службу; зіставити з відповідним підкодом T1053/T1547.

8	Privilege Escalation	Обрати файл, пов'язаний з обходом UAC чи експлуатацією підвищення привілеїв; перевірити рівень критичності (Level) спрацьованих правил.
9	Command and Control	Знайти приклад мережевої активності, що імітує C2-канал (маячки, DNS-тунелювання тощо); зіставити з тактикою Command and Control.
10	AutomatedTestingTools	Обрати файли, пов’язані з фреймворками автоматизованого red-team тестування.

Контрольні питання

1. Чим активний пошук загроз принципово відрізняється від реагування на алерти SIEM/IDS за фактом їхнього спрацювання?
2. Що таке Sigma-правило і чому воно є вендоронезалежним форматом опису детекції?
3. Чому в цій роботі використано публічний репозиторій реальних історичних EVTX-зразків, а не онлайн-SOC-платформа для дистанційного навчання?
4. Поясніть на власному досвіді, чому вивід MitreTactics/MitreTags міг бути порожнім за замовчуванням і що виправило ситуацію.
5. Чому одне й те саме правило Sigma («Log Cleared») спрацьовувало у файлі з розвідувальною активністю (BloodHound), хоча сама розвідка формально лишалася непокритою жодним готовим правилом?
6. Які два обов’язкові поля метаданих правила виявилися специфічною вимогою саме Hayabusa (а не загальної специфікації Sigma), і як їх відсутність проявлялася технічно?
7. Опишіть послідовність дій, якою ви скористалися б, щоб самостійно виявити мережеву активність, не покриту жодним готовим правилом (на прикладі розвідки через іменовані канали SAMR/LSARPC).
8. Яка практична різниця між прогоном Hayabusa на одному файлі (`-f`) та на цілій теці (`-d`) і коли варто застосовувати кожен із варіантів?

ЛАБОРАТОРНА РОБОТА 4: Безпека мережевих функцій, віртуалізованих на основі NFV

Мета: ознайомитися з архітектурою NFV (Network Functions Virtualization), побудувавши ланцюжок віртуалізованих мережевих функцій на основі Docker-контейнерів; продемонструвати специфічну для NFV вразливість порушення цілісності ланцюжка та її усунення.

Теоретична частина

Традиційно еволюція мереж базувалася на спеціалізованому апаратному забезпеченні (міжмережеві екрани, балансувальники навантаження, системи виявлення вторгнень), що потребувало значних капітальних витрат та ускладнювало масштабування. У відповідь на ці виклики виникла концепція віртуалізації мережевих функцій.

NFV (*Network Functions Virtualization*) – це архітектурний підхід, що передбачає віртуалізацію традиційних мережевих функцій та їх розгортання як програмних додатків на стандартних серверах загального призначення.

Згідно зі стандартами, архітектура NFV складається з трьох основних блоків:

— VNF (*Virtualised Network Functions*): мережеві функції, що традиційно реалізовувалися апаратно, тепер виконуються як програмні компоненти. VNF може складатися з однієї або кількох віртуальних машин чи контейнерів;

— NFVI (*NFV Infrastructure*): загальне, невиділене обчислювальне обладнання, яке забезпечує ресурси (обчислювальні, мережеві, дискові) для розгортання VNF. NFVI включає апаратні ресурси, віртуалізаційний шар та мережеву інфраструктуру.

— MANO (*Management and Orchestration*): підсистема управління, що здійснює оркестрацію сервісів та управління життєвим циклом VNF (розгортання, масштабування, видалення).

Ланцюжкове об’єднання службових функцій SFC (*Service Function Chaining*) – це впорядкована послідовність VNF, через яку повинен пройти певний клас трафіку (рис. 4.1). Наприклад, увесь вхідний веб-трафік має послідовно пройти через файрвол, систему виявлення вторгнень і балансувальник навантаження, перш ніж досягти застосунку.

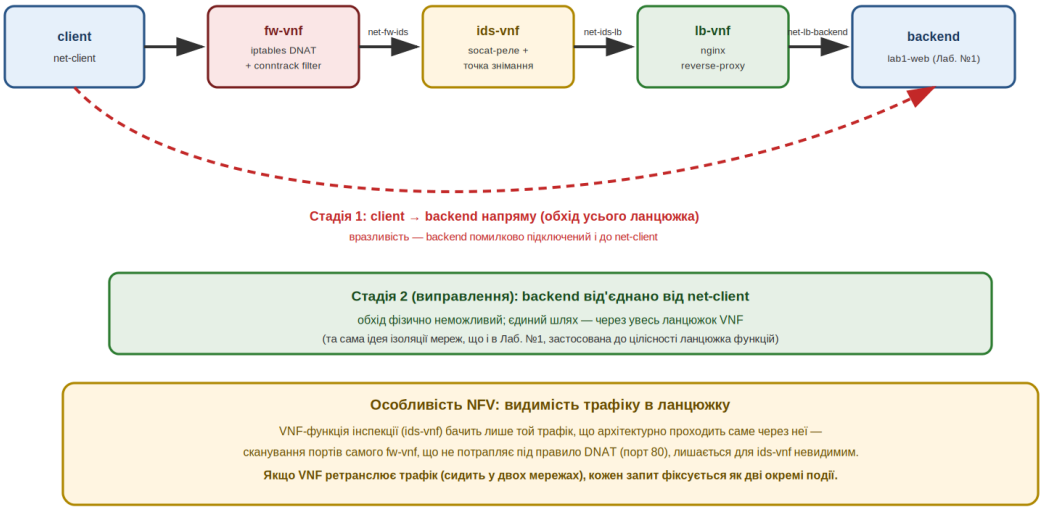


Рис. 4.1. Ланцюжок віртуалізованих мережевих функцій і вразливість його обходу

Специфічна для NFV категорія вразливостей пов’язана саме з гарантуванням цілісності такого ланцюжка. Якщо мережева топологія дозволяє трафіку досягти кінцевого застосунку в обхід одного чи кількох VNF, самé існування ланцюжка втрачає сенс незалежно від того, наскільки якісно реалізована кожна окрема функція. Крім того, компрометація однієї VNF у ланцюжку може дозволити зловмиснику обійти захисні механізми наступних функцій.

Перехід до віртуалізованих функцій створює нові вектори атак на різних рівнях архітектури: по-перше, проблема ізоляції інфраструктури NFVI, яка пов’язана з потенційною можливістю зловмисника «втекти» з контейнера чи віртуальної машини та отримати доступ до базового апаратного забезпечення або інших VNFs у випадку вразливості гіпервізора або

контейнерної платформи; по-друге, безпека оркестратора MANO, компрометація його (наприклад, через ненадійні API або крадіжку облікових даних) за наслідками порівняння з компрометацією централізованого SDN-контролера; і, по-третє, вразливості програмного забезпечення VNF.

У цій лабораторній роботі кожна VNF реалізована окремим Docker-контейнером, що є спрощеною, але архітектурно коректною моделлю NFVI: кожен контейнер є ізольованим, незалежно розгорнутим і замінюваним носієм однієї мережевої функції. Усі контейнери з'єднані поміж собою виключно через чітко означені мережеві інтерфейси, так само як реальні VNF з'єднуються через віртуальні комутатори у повноцінному середовищі NFVI.

Практична частина

Робота виконується на одному Windows-хості з Docker Desktop.

1. Побудова ланцюжка VNF:

1.1. Створити чотири окремі мережі (по одній для кожної ланки ланцюжка)

```
docker network create net-client
docker network create net-fw-ids
docker network create net-ids-lb
docker network create net-lb-backend
```

1.2. Підняти контейнер fw-vnf (Alpine + iptables), функція якого файрвол та NAT. Контейнер підключений одразу до мережі клієнта й до наступної ланки ланцюжка.

1.2.1. Створити Dockerfile

```
FROM alpine
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
RUN apk add --no-cache iptables bash
CMD ["sh", "-c", "sleep infinity"]
```

1.2.2. Створити і під'єднати fw-vnf до ланцюжка

```
docker build -t nfv-fw C:\LABs\2\fw
docker run -d --name fw-vnf --network net-client
--cap-add=NET_ADMIN --sysctl net.ipv4.ip_forward=1 nfv-fw
docker network connect net-fw-ids fw-vnf
```

Зауваження: спроба ввімкнути пересилання пакетів командою `echo 1 > /proc/sys/net/ipv4/ip_forward` всередині вже запущеного контейнера завершується помилкою «Read-only file system» оскільки тека /proc/sys змонтована в контейнері лише для читання, і навіть право NET_ADMIN цього не змінює. Параметр потрібно задавати ще на етапі `docker run` через прапорець `--sysctl` (як зазначено вище), змінити його для вже створеного контейнера неможливо, лише перестворивши контейнер із цим прапорцем.

1.3. Підняти контейнер ids-vnf, який відтворює мінімальну функцію інспекції (наразі TCP-реле на socat).

1.3.1. Створити Dockerfile

```
FROM alpine
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
RUN apk add --no-cache socat
CMD ["socat", "TCP-LISTEN:80,fork,reuseaddr", "TCP:lb-vnf:80"]
```

1.3.2. Створити і під’єднати ids-vnf до ланцюжка

```
docker build -t nfv-ids C:\LABs\2\ids
docker run -d --name ids-vnf --network net-fw-ids nfv-ids
docker network connect net-ids-lb ids-vnf
```

1.4. Побудувати й підняти контейнер lb-vnf (nginx reverse-proxy).

1.4.1. Створити Dockerfile

```
FROM nginx:alpine
ENV LC_ALL=C.UTF-8
ENV LANG=C.UTF-8
RUN apk add --no-cache bash
COPY nginx.conf /etc/nginx/nginx.conf
```

1.4.2. Створити nginx.conf

```
events {}
http {
```

```
    resolver 127.0.0.11 valid=10s;
    server {
        listen 80;
        location / {
            set $upstream_backend backend;
            proxy_pass http://$upstream_backend:80;
        }
    }
}
```

1.4.3. Створити і під’єднати lb-vnf до ланцюжка

```
docker build -t nfv-lb C:\LABs\2\lb
docker run -d --name lb-vnf --network net-ids-lb nfv-lb
docker network connect net-lb-backend lb-vnf
```

1.5. Підняти контейнер backend (повторне використання образу lab1-web). Навмисно підключити backend одразу і до net-lb-backend (правильна, кінцева ланка ланцюжка), і до net-client (навмисна помилка конфігурації для демонстрації вразливості на Стадії 1).

```
docker run -d --name backend --network net-lb-backend lab1-web
docker network connect net-client backend
```

1.6. Підняти клієнта.

```
docker run -d --name client --network net-client
nicolaka/netshoot sleep infinity
```

2. Демонстрація вразливості обходу ланцюжка:

2.1. Перевірити прямий доступ клієнта до backend в обхід усього ланцюжка VNF.

```
docker exec client curl -s http://backend
```

Результат: запит успішно повертає сторінку lab1-web попри існування повного ланцюжка VNF – клієнт досягає застосунку напряму, оминаючи fw-vnf, ids-vnf і lb-vnf. Це і є специфічна для NFV вразливість: наявність ланцюжка функцій нічого не варта, якщо мережева топологія дозволяє його обійти.

3. Налаштування легітимного шляху через fw-vnf:

3.1. Визначити реальні назви мережевих інтерфейсів fw-vnf (порядок eth0/eth1 відповідає порядку підключення мереж при створенні контейнера)

```
docker exec fw-vnf ip -4 addr
```

3.2. Прописати DNAT (переспрямування вхідного трафіку на порт 80 до ids-vnf), MASQUERADE (підміна зворотної адреси) і базову фільтрацію за портом.

```
docker exec fw-vnf sh -c "IDS_IP=$(getent hosts ids-vnf | awk
'{print $1}'); iptables -t nat -A PREROUTING -i eth0 -p tcp
--dport 80 -j DNAT --to-destination ${IDS_IP}:80; iptables -t
nat -A POSTROUTING -o eth1 -j MASQUERADE; iptables -A FORWARD -p
tcp --dport 80 -j ACCEPT; iptables -A FORWARD -j DROP"
docker exec fw-vnf iptables -I FORWARD 1 -m conntrack --ctstate
ESTABLISHED,RELATED -j ACCEPT
```

3.3. Перевірити прохід легітимного трафіку через увесь ланцюжок

```
docker exec client curl -s http://fw-vnf
```

```
PS C:\WINDOWS\system32> docker exec client curl -s http://fw-vnf
<h1>Corporate Web Portal</h1><p>Status: OK</p>
```

Рис. 4.2. Проходження легітимного трафіку через ланцюжок VNF

4. Усунення вразливості

4.1. Від’єднати backend від мережі клієнта, залишивши лише зв'язок з lb-vnf

```
docker network disconnect net-client backend
```

4.2. Повторно перевірити обидва шляхи

```
docker exec client curl -s --max-time 3 http://backend
docker exec client curl -s http://fw-vnf
```

Результат: пряме звернення до backend завершується тайм-аутом (обхід усунений – фізично відсутній мережевий шлях), тоді як звернення через fw-vnf, як і раніше, успішно повертає сторінку сервісу. Вразливість порушення цілісності ланцюжка виправлена тим самим принципом ізоляції мереж, що застосовувався у лабораторній роботі №1, але тепер спрямованим на захист самої структури ланцюжка функцій, а не лише сегментів мережі.

5. Підключення функції інспекції трафіку (Suricata) до вузла ids-vnf:

5.1. Підключити тимчасовий sniffer-контейнер до мережевого простору ids-vnf і згенерувати трафік.

```
docker run -d --name nfv-sniffer --network container:ids-vnf
nicolaka/netshoot tcpdump -i any -w /capture.pcap
Start-Sleep -Seconds 2
for ($i=0; $i -lt 10; $i++) { docker exec client curl -s
--max-time 1 http://fw-vnf | Out-Null }
Start-Sleep -Seconds 2
docker stop nfv-sniffer
docker cp nfv-sniffer:/capture.pcap C:\LABs\2\ids_capture.pcap
docker rm nfv-sniffer
```

Зауваження 1: початкова спроба згенерувати підозрілу активність командою `nmap -sS -p 1-200 fw-vnf` не дала жодного результату при подальшому аналізі – захоплення показало лише 59 пакетів замість очікуваних сотень. Причина: сканування було спрямоване на порти самого fw-vnf, тоді як правило DNAT переспрямовує до ids-vnf лише трафік на порт 80, решта 199 портів обробляється власним ланцюжком INPUT маршрутизатора fw-vnf і ніколи не перетинає сегмент, де підключений sniffer. Це важливий, специфічний для NFV висновок: функція інспекції бачить лише той трафік, який архітектурно проходить саме через неї (на відміну від традиційного мережевого IDS на дзеркалі порту, що бачить увесь трафік сегмента). Рішенням цієї проблеми є генерація повторюваних легітимних запитів на порт 80 (показано вище), а не сканування довільних портів.

Зауваження 2: перша спроба зупинити sniffer одразу після генерації трафіку (без паузи) дала порожній або пошкоджений pcap-файл (Suricata: «pcap_next_ex(): -2», «failed to get first packet timestamp»), оскільки tcpdump не встиг записати захоплені пакети на диск до отримання сигналу зупинки. Рішенням цієї проблеми є невеликі паузи (Start-Sleep) до й після генерації трафіку та перевірка docker logs sniffer-контейнера перед зупиненням.

5.2. Проаналізувати захоплення офлайн через Suricata з власним правилом виявлення сплеску TCP-з’єднань.

5.2.1. Створити власне правило local.rules для Suricata

```
alert tcp any any -> any any (msg:"NFV-Lab: TCP SYN scan detected"; flags:S; threshold: type both, track by_src, count 5, seconds 3; sid:1000001; rev:1;)
```

5.2.2. Проаналізувати дамп трафіку ids_capture.pcap

```
docker run --rm -v C:\LABs\2:/pcaps -v
C:\LABs\2\suricata-output:/output -v
C:\LABs\2\rules:/etc/suricata/rules jasonish/suricata -r
/pcaps/ids_capture.pcap -l /output -S
/etc/suricata/rules/local.rules -k none
type C:\LABs\2\suricata-output\eve.json | findstr "NFV-Lab"
```

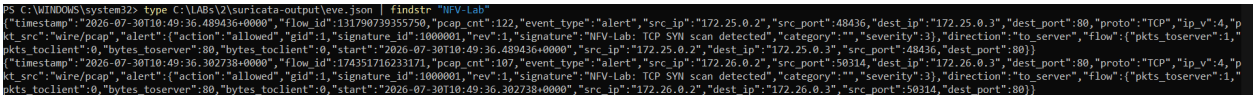


Рис. 4.3. Результати аналізу дампу трафіку

Результат: правило спрацювало, але дало ДВІ окремі події тривоги замість однієї з різними парами адрес: 172.25.x.x → 172.25.x.x і 172.26.x.x → 172.26.x.x. Причина цього наступна: ids-vnf одночасно перебуває у двох мережах (net-fw-ids і net-ids-lb), оскільки виконує ретрансляцію трафіку далі по ланцюжку. Захоплення `-i any` на цьому вузлі фіксує ОБИДВА «стрибки» одного логічного запиту клієнта – вхідний (від fw-vnf) і вихідний (до lb-vnf) – як два незалежні TCP-потоки. Це ключовий методологічний висновок для аналізу трафіку в SFC: якщо VNF виконує проксіювання/ретрансляцію, кожна подія на ній потенційно подвоюється, і аналітик повинен враховувати цю топологічну особливість, щоб не порахувати одну реальну подію як дві незалежні.

Індивідуальні завдання

Кожен здобувач отримує індивідуальний сценарій (табл. 4.1) із власною додатковою VNF-функцією, яку потрібно вбудувати в ланцюжок (додатково до базових fw-vnf/ids-vnf/lb-vnf). Завдання:

— реалізувати додаткову VNF-функцію окремим Docker-контейнером і вставити її в ланцюжок у логічно доречному місці;

— відтворити демонстрацію вразливості обходу ланцюжка та її усунення для своєї топології;

— згенерувати трафік, що демонструє роботу саме доданої функції (за таблицею);

— підключити sniffer до нового вузла й підтвердити захоплення трафіку, специфічного для цієї функції;

— оформити звіт зі скріншотами кожного етапу.

Таблиця 4.1. Варіанти індивідуальних завдань

№	Сценарій	Додаткова VNF-функція в ланцюжку	Трафік для перевірки виявлення
1	Інтернет-магазин	waf-vnf – nginx з правилом фільтрації запитів, що містять шаблон SQL-ін’єкції в URI	curl із рядком '=1 OR 1=1' у параметрі запиту
2	Банківське відділення	antifraud-vnf – мінімальний Flask-мок, що повертає Risk-Score за IP джерела	серія запитів з однієї адреси за короткий час
3	Телеком-оператор	qos-vnf – обмеження пропускної здатності через tc (traffic control) на вихідному інтерфейсі	паралельні запити для перевірки застосованого обмеження
4	Державний портал	dlp-vnf – nginx з блокуванням тіла запиту, що містить шаблон номера документа (наприклад, серію й номер паспорта)	POST-запит із тестовим номером документа в тілі
5	Медична установа	anonymizer-vnf – простий проксі-скрипт, що маскує значення визначеного поля у відповіді backend	запит, що повертає дані з тестовим ідентифікатором пацієнта
6	Промислове підприємство (IT/OT)	protocol-gateway-vnf – контейнер, що приймає трафік на нестандартному порту і транслює його до backend (імітація шлюзу Modbus/OPC-UA)	запит на нестандартний порт шлюзу

Контрольні питання

1. Що таке NFV і чим віртуалізована мережева функція відрізняється від традиційного апаратного мережевого пристрою?
2. Що таке Service Function Chaining і чому порядок проходження трафіку через ланцюжок VNF має бути гарантований архітектурно, а не лише за домовленістю?
3. Опишіть конкретну вразливість, продемонстровану в цій роботі (обхід ланцюжка VNF), і поясніть, чому вона є специфічною саме для NFV, а не загальною мережевою проблемою.
4. Чому спроба виконати `echo 1 > /proc/sys/net/ipv4/ip_forward` всередині запущеного контейнера завершилась помилкою «Read-only file system», і як прапорець `--sysctl` при docker run вирішує цю проблему?
5. Поясніть, чому правило iptables, що дозволяє пересилання лише за портом призначення 80, заблокувало легітимне з'єднання. Яку роль відіграє відстеження з'єднань (conntrack, стан ESTABLISHED/RELATED) у стандартному, коректному файрволі?
6. Чому спроба просканувати порти самого fw-vnf інструментом nmap не була зафіксована функцією інспекції трафіку (ids-vnf), хоча сканування явно було спрямоване на вузол ланцюжка?
7. Поясніть, чому одна й та сама подія (один запит клієнта) була зафіксована Suricata як дві окремі підозрілі події при аналізі трафіку, знятого з вузла ids-vnf?
8. Які паралелі можна провести між атаками на SDN-контролер (централізована площина керування) і потенційними атаками на оркестратор NFV (MANO)?
9. Чому демонстрація вразливості порушення цілісності ланцюжка VNF в цій роботі є прямим наслідком помилки конфігурації мережі (backend, підключений одразу до двох сегментів), а не недоліком самих VNF-функцій?

10. Наведіть приклад, чим ізоляція між орендарями/функціями (multi-tenancy) на спільній NFV-інфраструктурі відрізняється від традиційної ізоляції на виділених апаратних пристроях, і які нові ризики це створює.

ЛАБОРАТОРНА РОБОТА 5: Аналіз безпечних протоколів нового покоління

Мета: навчитися аналізувати структуру мережевих пакетів сучасних протоколів шифрування трафіку (DoH, DoT, QUIC, WireGuard) за допомогою програмного комплексу Wireshark; зрозуміти, який саме рівень інформації приховує кожен протокол, а який — залишається доступним для пасивного атакуючого.

Теоретична частина

Традиційні засоби мережевої безпеки (IDS/IPS, DPI тощо) історично покладалися на можливість прочитати два джерела метаданих у відкритому вигляді: DNS-запит (яке ім’я резолвиться) і поле SNI у TLS Client Hello (до якого домену встановлюється з’єднання). Протоколи, які будуть розглянуті в цій роботі, послідовно закривають кожен із цих каналів витоку – але, як буде показано практично, кожен закриває лише певний, чітко обмежений рівень, а не весь ланцюжок одразу.

Інкапсуляція DNS-запиту: протоколи DoH / DoT. DNS over HTTPS (RFC 8484) і DNS over TLS (RFC 7858) розв’язують одну й ту саму задачу різними транспортними засобами: замість того, щоб надсилати DNS-запит у відкритому вигляді (UDP/53), клієнт інкапсулює той самий запит (у стандартному DNS wire-форматі) усередину зашифрованого TLS-з'єднання – в DoT це «голий» TLS на порту 853, у DoH – той самий запит передається в тілі HTTP-запиту (GET/POST) поверх HTTPS на порту 443, що додатково маскує сам факт DNS-активності під звичайний веб-трафік. У перехопленому трафіку в обох випадках безпосередньо DNS-протокол (як окремий дисектор Wireshark) не з’являється – лише зашифрований TLS-потік до IP-адреси резолвера.

TLS Client Hello. Незалежно від того, DoH це, DoT чи звичайний HTTPS-запит до сайту, встановлення з’єднання починається з пакета Client Hello – першого повідомлення TLS-рукостискання, яке за

специфікацією передається без шифрування. Це повідомлення містить перелік розширень (*extensions*), серед яких – server_name (SNI): явна вказівка, до якого імені хоста клієнт хоче підключитися (це потрібно серверу для правильної маршрутизації запиту, якщо на одній IP-адресі розміщено кілька доменів). Саме це поле, попри повністю зашифрований DNS-запит через DoH, залишається читабельним для будь-кого, хто перехоплює трафік – просто відкривши дерево пакета в Wireshark і розгорнувши Handshake Protocol → Extension: server_name.

Encrypted Client Hello (ECH) вирішує саме проблему видимого SNI: Client Hello поділяється на дві частини – *Outer* (зовнішня, видима спостерігачу) та *Inner* (внутрішня, зашифрована через HPKE) (рис. 5.1). У видимій, зовнішній частині поле server_name замінюється на узагальнене «прикривне» ім'я хостинг-провайдера (наприклад, cloudflare-ech.com для сайтів під захистом Cloudflare) – саме це ім'я і бачить пасивний спостерігач, тоді як справжнє ім'я домену передається лише всередині зашифрованого блока Inner Client Hello, розшифрувати який може виключно сервер, що володіє відповідним приватним ключем ECH-конфігурації.

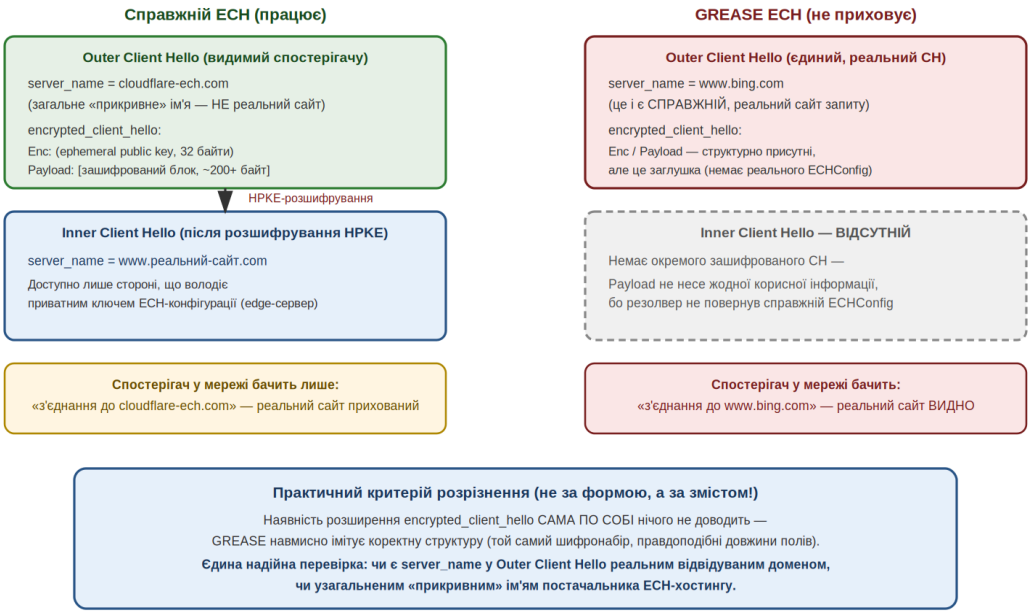


Рис. 5.1. Структура TLS Client Hello

Зауважимо, що публічний ключ та «прикривне» ім’я (Outer SNI) браузер дізнається ще до встановлення TLS-з’єднання – під час розв’язання DNS через спеціальні записи типу HTTPS (Type 65 / SVCB). Якщо DoH чи DoT вимкнені або перехоплюються, пасивний спостерігач може зафіксувати факт використання ECH та отримати конфігураційні параметри ще на етапі звичайного DNS-запиту.

Механізм GREASE ECH. Сучасні браузери додають розширення encrypted_client_hello до практично кожного TLS-з’єднання – навіть тоді, коли сервер узагалі не підтримує ECH і не публікував жодної ECH-конфігурації в DNS. Це навмисне архітектурне рішення – механізм GREASE (*Generate Random Extensions and Sustain Extensibility*): якщо браузери додавали б це розширення лише за наявності реальної підтримки з боку сервера, спостерігач у мережі міг би за самим фактом присутності розширення статистично визначати, які з’єднання захищені ECH, а які ні. Таким чином, додаючи структурно правдоподібну, але функціонально порожню (GREASE) версію розширення до кожного з’єднання, браузер робить усі TLS-сесії однаково виглядаючими ззовні – незалежно від того, чи є за ними реальний ECH.

Звідси випливають головні практичні висновки для мережевого аналізу:

1. Перевірка реального шифрування SNI. Сама наявність розширення encrypted_client_hello (з коректним шифронабором, полями Enc і Payload правдоподібної довжини) НЕ є доказом того, що ім’я домену реально приховане. Єдина надійна перевірка – подивитися, чим саме є значення server_name у видимому (Outer) Client Hello:
    - Якщо це узагальнене, «прикривне» ім’я постачальника (не пов’язане з фактично відвідуваним сайтом) – ECH справді працює.
    - Якщо ж там стоїть справжня, реальна адреса сайту, який відвідав користувач, – це означає GREASE-заглушку (сервер не надав робочої ECH-конфігурації для цього з'єднання), і жодного

приховування фактично не відбулося, попри зовні «серйозну» криптографічну структуру розширення.

2. Залишкові канали витоку (IP-адреса та ASN). Навіть за повного використання зв’язки DoH + ECH пасивний спостерігач усе одно бачить цільову IP-адресу та її належність до автономної системи (ASN). Приховування домену за допомогою ECH є ефективним лише тоді, коли цільовий ресурс розміщений на спільній CDN-інфраструктурі (наприклад, Cloudflare), де одна IP-адреса обслуговує тисячі різних сайтів. Якщо ж сайт використовує виділену IP-адресу, аналітик може легко визначити фактичний ресурс за допомогою IP-геолокації та Reverse DNS / PTR-запитів.

Протокол QUIC. QUIC (RFC 9000 / RFC 9001), який лежить в основі HTTP/3, працює поверх UDP, а не TCP. Це само по собі ускладнює роботу традиційних TCP-орієнтованих систем DPI/IDS, розрахованих на семантику SYN/ACK та послідовних номерів TCP-сегментів. Пакети QUIC поділяються на:

— Long Header, які використовуються під час встановлення з’єднання (типи Initial, Handshake, 0-RTT);

— Short Header, які використовуються після завершення рукостискання для передачі прикладних даних.

Принципово важлива деталь, що секретні ключі для захисту саме Initial-пакетів виводяться не з приватних ключів сторін з’єднання, а з публічно відомої, фіксованої константи (*salt*, визначеної в RFC 9001 для QUIC v1) у поєднанні з полем Destination Connection ID, яке передається в самому пакеті у відкритому вигляді. Це означає, що будь-яка сторона, яка знає цю публічну константу (зокрема й Wireshark «з коробки»), може обчислити ці ключі й розшифрувати вміст Initial-пакета без жодного секрету, невідомого сторонньому спостерігачу. Мета такого «шифрування» – не приховати вміст від спостерігача, а забезпечити цілісність і гнучкість версіонування протоколу на найпершому етапі з'єднання.

CRYPTO- і STREAM-фрейми. Усередині розшифрованого Initial-пакета передаються CRYPTO-фрейми. Вони переносять фрагменти повідомлень TLS-рукостискання (той самий Client Hello, з тим самим полем SNI, що й у звичайному TLS 1.3). Оскільки один пакет QUIC розміром близько 1200–1300 байт зазвичай замалий, щоб вмістити повний Client Hello з усіма розширеннями (обсяг якого може сягати 2000+ байт), Wireshark збирає кілька CRYPTO-фрагментів із послідовних пакетів у суцільний блок. У дереві пакета це відображається як позначка [Reassembled PDU in frame: N], перехід за якою показує повністю розібраний TLS Client Hello.

Таким чином, важливо не плутати CRYPTO- і STREAM-фреймами – останні з'являються лише після завершення рукостискання й переходу на application-рівень шифрування (захищений уже узгодженими сесійними ключами, недоступними пасивному спостерігачу). Саме тому функція Wireshark «Follow QUIC Stream» не знаходить нічого на Initial-пакетах: там технічно немає STREAM-даних, а лише CRYPTO-фрейми.

Протокол WireGuard. WireGuard – це протокол VPN на основі криптографічного фреймворку Noise Protocol, який (на відміну від TLS) не має окремого рукостискання із сертифікатами. Установлення сесійних ключів відбувається шляхом обміну короткими повідомленнями Handshake Initiation та Handshake Response на основі заздалегідь обмінюваних статичних публічних ключів сторін. Після завершення рукостискання весь подальший трафік передається у вигляді пакетів Transport Data:

— *Відкрита частина* містить лише службові заголовочні поля (ідентифікатор одержувача сесії та лічильник пакетів);

— *Зашифрована частина* (ChaCha20-Poly1305) містить цілісний оригінальний IP-пакет, включно з його власними IP/TCP/UDP-заголовками, а не лише корисне навантаження застосунку.

Принципова відмінність від TLS/HTTPS полягає в тому, що у TLS шифрується лише вміст з'єднання (*application layer*), тоді як IP/TCP-заголовки транспортного рівня залишаються відкритими для

будь-кого в мережі. WireGuard повністю ховає внутрішню мережеву структуру усередині зашифрованого контейнера. Зважаючи на таку архітектуру, перехоплення трафіку між двома кінцевими точками WireGuard-тунелю показує виключно зашифровані UDP-пакети (за замовчуванням на порту UDP 51820). Без наявності приватних ключів сесії розпізнати протокол, порт чи IP-адресу оригінального (інкапсульованого) пакета неможливо.

Побачити оригінальний, розшифрований трафік у Wireshark або tcpdump можливо лише в місцях, де він існує у відкритому вигляді – на віртуальному мережевому інтерфейсі (wg0) однієї з кінцевих точок тунелю або у мережі за шлюзом, якщо WireGuard розгорнуто в топології «шлюз–шлюз».

Таблиця 5.1. Порівняння розглянутих протоколів з точки зору видимості метаданих для засобів мережевого моніторингу

Протокол	**Що приховується**	**Що залишається видимим**
DoH / DoT	Вміст DNS-запиту/відповіді (яке ім’я резолвиться)	Факт з’єднання із самим DoH/DoT-резолвером (SNI резолвера); DNS-трафік інших застосунків, що не використовують цей резолвер
TLS SNI (без ECH)	Вміст HTTP-запиту, сертифікат	Ім’я домену призначення (server_name) – читається відкритим текстом у Client Hello; IP-адреса призначення
ECH (справжній)	Реальне ім’я домену (Inner SNI)	Лише прикривне ім’я хостинг-провайдера (Outer SNI); IP-адреса призначення
QUIC (Initial)	Нічого – Initial-пакет розшифровується публічними ключами за специфікацією	Той самий Client Hello/SNI, що й у звичайному TLS – захист лише номінальний
WireGuard	Увесь вміст оригінального пакета (включно з заголовками IP, ICMP тощо)	Факт з’єднання між двома IP-адресами кінцевих точок тунелю на порту WireGuard (UDP)

Підсумовуючи розглянуті технології шифрування та інкапсуляції, варто зазначити, що жоден із протоколів (за винятком повноцінних VPN-рішень) не забезпечує абсолютної анонімності на мережевому рівні. Кожен інструмент

вирішує свою вузьку задачу та залишає певні метадані відкритими для пасивного спостерігача або систем DPI. Для зручності систематизації цих особливостей, у табл. 5.1 наведено зведене порівняння того, що саме приховує кожен протокол, а що залишається доступним для аналізу.

Практична частина

Робота виконується на одному Windows-хості з використанням двох програмним комплексів: Wireshark (для перехоплення й аналізу трафіку) та Docker Desktop (для побудови WireGuard-тунелю в ізольованих контейнерах).

Частина А. Аналіз DNS over HTTPS і QUIC засобами Wireshark

1. Виконати контрольний (базовий) захват:

1.1. Переконатися, що DoH у браузері вимкнено.

1.2. Відкрити Wireshark на основному мережевому адаптері (Wi-Fi / Ethernet) та почати захоплення трафіку.

1.3. У браузері відкрити 2-3 нові сайти (бажано такі, які не відвідувалися довгий час, задля запобігання використанню кеша DNS).

1.4. Зупинити захоплення трафіку і зберегти файл (baseline_plain_dns.pcapng).

1.5. Проаналізувати отриманий файл, застосувавши фільтр `dns` і переконатися, що DNS-запити/відповіді читаються відкритим текстом (доменні імена видно прямо в колонці Info) (рис. 5.2).

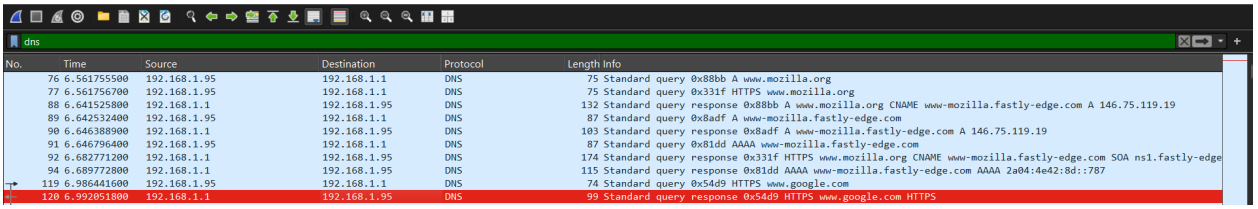


Рис. 5.2. Аналіз захопленого трафіку у випадку вимкненого DoH

2. Увімкнути DoH у браузері[3]:

[3] Для Firefox: Settings → Privacy & Security → DNS over HTTPS → Max Protection;
Для Chrome / Edge: Settings → Privacy and security → Security → Use secure DNS

2.1. Виконати новий захват трафіку, відвідавши ті самі сайти, і зберегти файл (doh_enabled.pcapng).

2.2. Застосувати фільтр `dns` до нового захоплення. Звернути увагу: записи DNS можуть не зникнути повністю – фоновий трафік інших застосунків (антивірус, службові процеси ОС, офісні пакети тощо) продовжує резолвити імена через звичайний системний DNS-стек, оскільки налаштування DoH у браузері діють лише для запитів самого браузера, а не для всієї операційної системи. Перевірити точково: чи є серед записів dns саме ті домени сайтів, які навмисно відвідувалися на кроці 2 (фільтр `dns.qry.name contains "<частина домену>"`) – їх не повинно бути.

2.3. Застосувати фільтр `tls.handshake.extensions_server_name` до того самого захоплення. Знайти сесії до відвіданих сайтів і переконатися, що поле `server_name` у Client Hello читається відкритим текстом, попри увімкнений DoH, що підтверджує, що DoH ховає лише сам DNS-запит, а не факт з'єднання із сайтом (рис. 5.3).

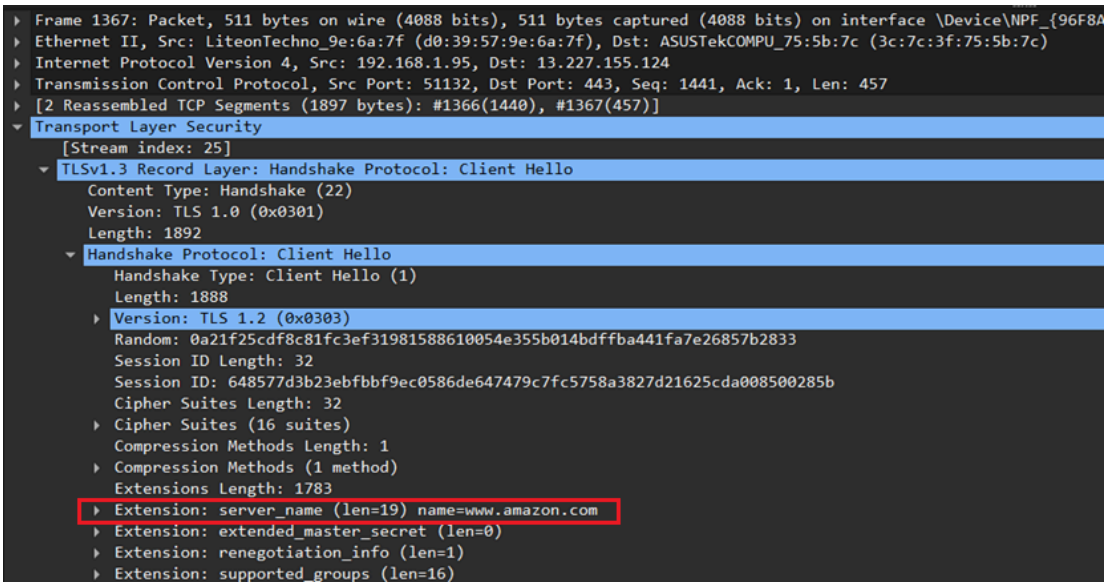


Рис. 5.3. Поле `server_name` у Client Hello у випадку увімкненого DoH

2.4. Знайти в тому самому захопленні сесію безпосередньо до DoH-резолвера (наприклад, фільтр `tls.handshake.extensions_server_name contains`

"cloudflare-ech") – це єдиний TLS-потік, що відповідає за саму приховану функцію резолвінгу (рис. 5.4).

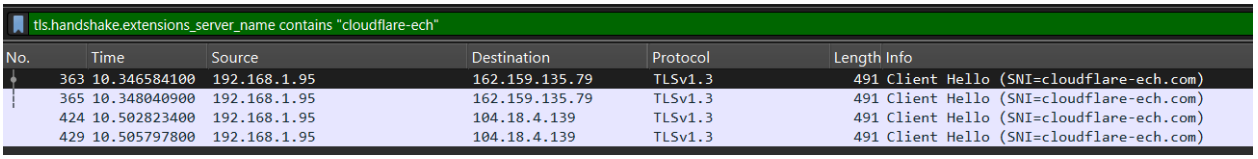


Рис. 5.4. Захоплена сесія до DoH-резолвера

3. Проаналізувати механізм ECH:

3.1. Застосувати фільтр `tls.handshake.extension.type == 65037` – звернути увагу, що більшість сучасних сесій браузера містять це розширення.

3.2. Розгорнути кілька випадкових записів у Packet Details (Handshake Protocol → Extension: encrypted_client_hello) і зафіксувати для кожного: значення `server_name` у Outer Client Hello, поле Client Hello type (Outer/Inner), шифронабір (Cipher Suite), довжини полів Enc і Payload (рис. 5.5).

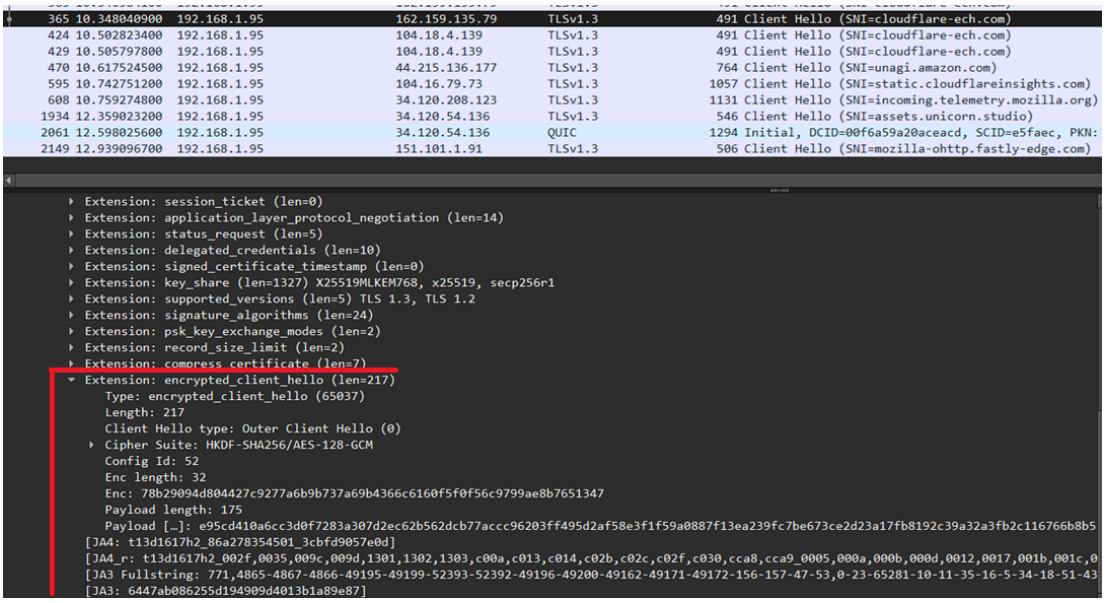


Рис. 5.5. Розгорнута вкладка encrypted_client_hello

3.3. Визначити для кожного знайденого прикладу, чи це справжній ECH, чи GREASE-заглушка, застосовуючи практичний критерій із теоретичної частини: якщо `server_name` – узагальнене ім'я хостинг-провайдера

(наприклад, cloudflare-ech.com) – це працюючий ECH; якщо там реальна адреса відвідуваного сайту – це GREASE. Знайти щонайменше по одному прикладу кожного випадку і зафіксувати скріншотами розгорнутої структури обох.

4. Проаналізувати протокол QUIC:

4.1. Застосувати фільтр `quic.long.packet_type == 0` (Initial-пакети) до того самого чи нового захоплення (бажано відвідати сайт, що використовує HTTP/3 – більшість сервісів Google чи Cloudflare).

4.2. Знайти в дереві пакета позначку [Reassembled PDU in frame: N] (рис. 5.6), перейти за нею й переконатися, що там розгортається повне дерево TLSv1.3 Record Layer → Handshake Protocol: Client Hello з тим самим полем `server_name` (рис. 5.7).

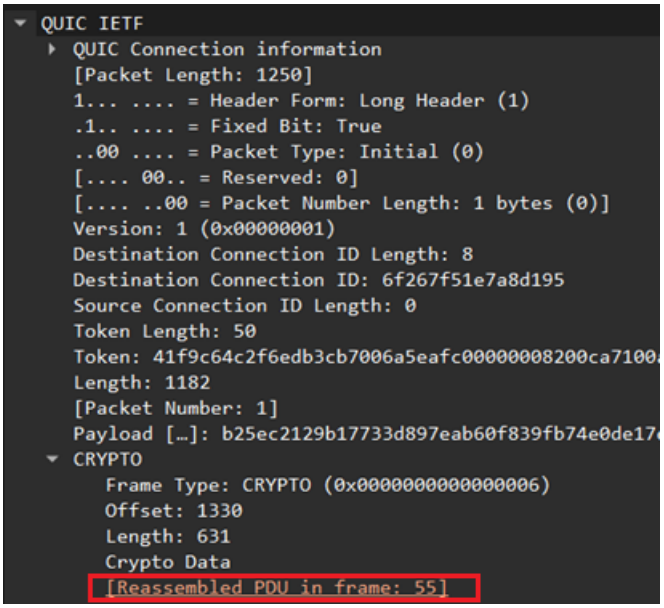


Рис. 5.6. Позначка «Reassembled PDU in frame: 55»

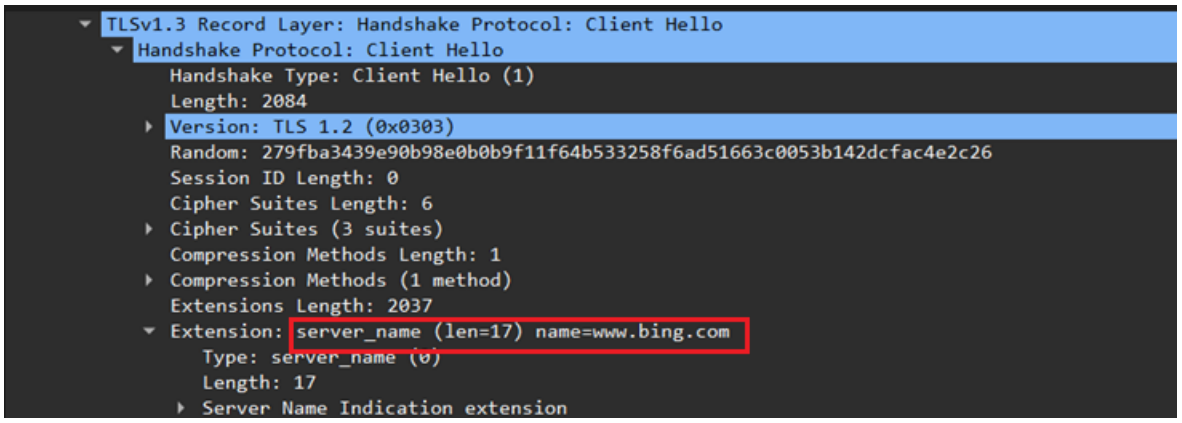


Рис. 5.7. Client Hello з `server_name` www.bing.com

4.3. Спробувати застосувати Follow → QUIC Stream до цього ж Initial-пакета й переконатися, що з'являється повідомлення «QUIC stream not found» — пояснити в звіті, чому це очікувана поведінка (CRYPTO-фрейм – не STREAM-фрейм, прикладні дані ще не передаються на етапі Initial).

Примітка щодо DoT: аналіз DoT свідомо не входить до обов'язкової частини цієї роботи, оскільки він ілюструє ту саму тезу, що й DoH (шифрування DNS-запиту через TLS), лише іншим транспортом (безпосередньо TLS на порту 853 замість HTTPS). Перевірка DoT входить до індивідуального завдання.

Частина Б. Побудова та аналіз інкапсуляції WireGuard-тунелю

1. Створити ізольовану Docker-мережу та підняти два контейнери з WireGuard (рис. 5.8):

```
docker network create wgnet
docker run -d --name wg-a --network wgnet --cap-add=NET_ADMIN
--cap-add=SYS_MODULE --device=/dev/net/tun -e PUID=1000 -e
PGID=1000 -e TZ=Europe/Kyiv linuxserver/wireguard
docker run -d --name wg-b --network wgnet --cap-add=NET_ADMIN
--cap-add=SYS_MODULE --device=/dev/net/tun -e PUID=1000 -e
PGID=1000 -e TZ=Europe/Kyiv linuxserver/wireguard
```

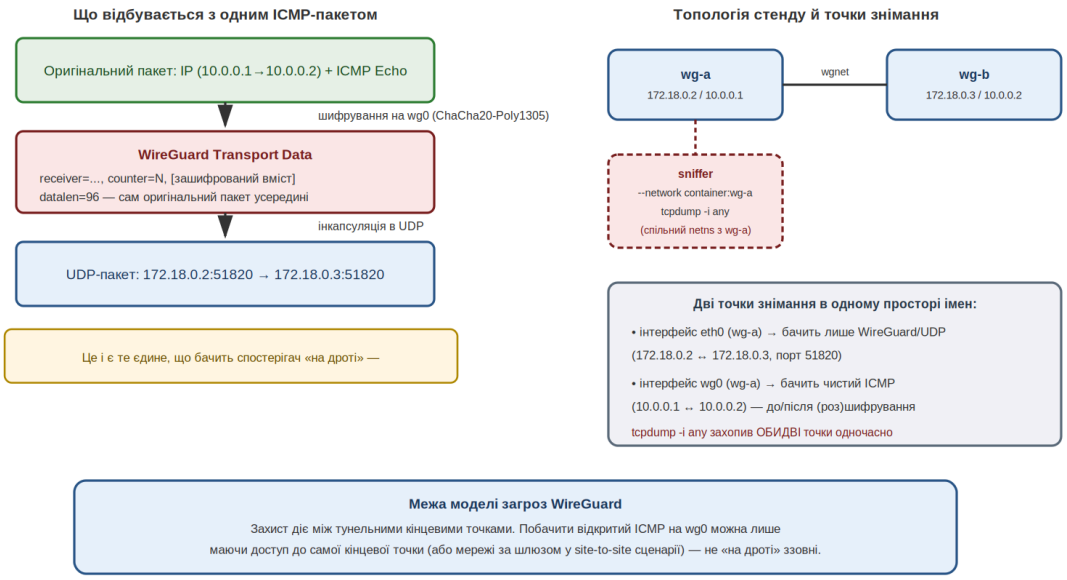


Рис. 5.8. Інкапсуляція трафіку WireGuard і точки знімання стенду

2. Виконати контрольний (незашифрований) захват трафіку (ICMP-трафік між піднятими контейнерами) «до» налаштування тунелю і зберегти дамп у файл wg_before_tunnel.pcapng. Оскільки Docker Desktop на Windows виконує самі контейнери всередині внутрішньої Linux-VM (WSL2), інтерфейси Windows (vEthernet) не бачать трафік між контейнерами напряму, тому захоплення виконується тимчасовим контейнером-«сніффером», що ділить мережевий простір імен із wg-a.

```
docker run -d --name sniffer --network container:wg-a
nicolaka/netshoot tcpdump -i any -w /capture.pcap icmp
docker exec -it wg-a ping -c 4 wg-b
docker stop sniffer
docker cp sniffer:/capture.pcap
C:\LABs\5\wg_before_tunnel.pcapng
docker rm sniffer
```

3. Відкрити wg_before_tunnel.pcapng у Wireshark, застосувати фільтр `icmp`, переконатися, що видно повністю відкриті ICMP Echo request/reply з читабельними полями ID/SEQ/TTL. Зберегти скріншот як контрольний доказ «до» (рис. 5.9).

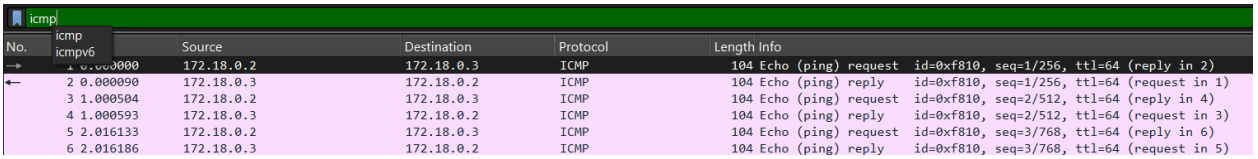


Рис. 5.9. Приклад дампу трафіка з читабельними полями ID/SEQ/TTL

4. Згенерувати ключі на обох контейнерах.

```
docker exec wg-a sh -c "wg genkey | tee /config/privatekey-a |
wg pubkey > /config/publickey-a; cat /config/publickey-a"
docker exec wg-b sh -c "wg genkey | tee /config/privatekey-b |
wg pubkey > /config/publickey-b; cat /config/publickey-b"
docker exec wg-a cat /config/privatekey-a
docker exec wg-b cat /config/privatekey-b
```

5. Створити конфігураційні файли wg0.conf для обох контейнерів (підставивши реальні значення отриманих ключів) у вигляді текстових файлів на Windows-хості й скопіювати їх у контейнери. Для контейнеру A wg0-a.conf:

```
@"
[Interface]
PrivateKey = <ПРИВАТНИЙ_КЛЮЧ_A>
Address = 10.0.0.1/24
ListenPort = 51820

[Peer]
PublicKey = <ПУБЛІЧНИЙ_КЛЮЧ_B>
AllowedIPs = 10.0.0.2/32
Endpoint = wg-b:51820
PersistentKeepalive = 25
"@ | Set-Content -NoNewline -Encoding ASCII wg0-a.conf
docker cp wg0-a.conf wg-a:/config/wg0.conf
```

Аналогічно створити й скопіювати другий конфіг (wg0-b.conf) для wg-b, дзеркально помінявши місцями ключі, адреси (10.0.0.2/24) та Endpoint (wg-a:51820).

6. Перезапустити обидва контейнери, щоб застосувати нову конфігурацію, і перевірити стан WireGuard-інтерфейсу.

```
docker restart wg-a; docker restart wg-b
docker exec wg-a wg show
```

У виводі команди має з’явитися активний peer із заповненим полем latest handshake (означає успішне встановлення сесії) і зростаючим лічильником transfer.

7. Виконати ping-запит через WireGuard-адреси (не через Docker-адреси контейнерів) і одночасно зняти новий захват тим самим способом (тимчасовий sniffer-контейнер, `--network container:wg-a`, `tcpdump -i any` без обмеження за ICMP – навмисно, щоб побачити обидва рівні одночасно).

```
docker run -d --name sniffer2 --network container:wg-a
nicolaka/netshoot tcpdump -i any -w /capture2.pcap
docker exec -it wg-a ping -c 4 10.0.0.2
docker stop sniffer2
docker cp sniffer2:/capture2.pcap
C:\LABs\5\wg_after_tunnel.pcapng
docker rm sniffer2
```

8. Відкрити файл дампу wg_after_tunnel.pcapng у Wireshark і застосувати по черзі два фільтри для розділення двох точок знімання:

- `ip.addr == 172.18.0.2 or ip.addr == 172.18.0.3` (зовнішній, зашифрований вигляд: протокол WireGuard, поля Handshake, Keepalive, Transport Data, вміст нерозбірний) (рис. 5.10),
- `ip.addr == 10.0.0.1 or ip.addr == 10.0.0.2` (внутрішній вигляд, знятий на інтерфейсі wg0: чистий, розшифрований ICMP) (рис. 5.11).

Зробити по одному скріншоту для кожного фільтра.

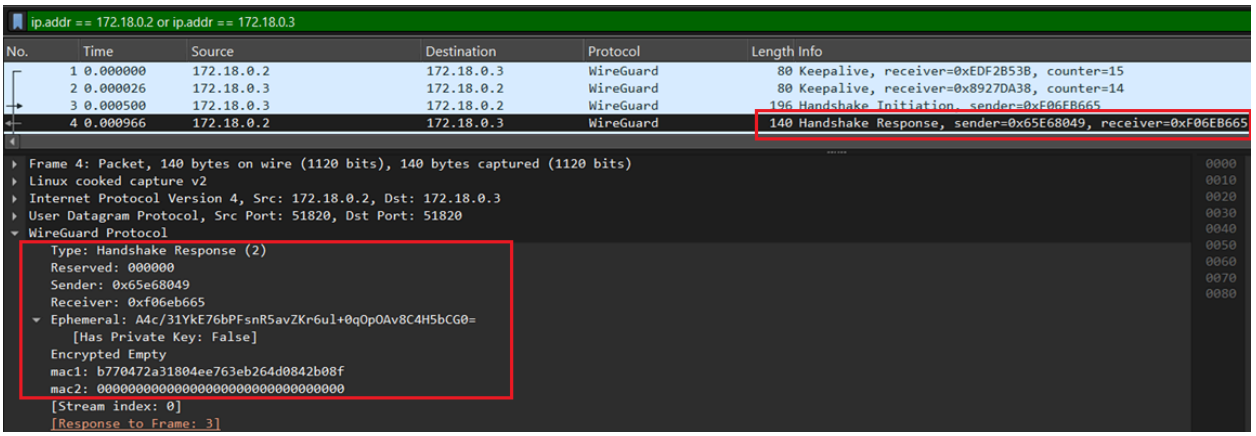


Рис. 5.10. Зовнішній, зашифрований вигляд при використанні фільтру `ip.addr == 172.18.0.2 or ip.addr == 172.18.0.3`

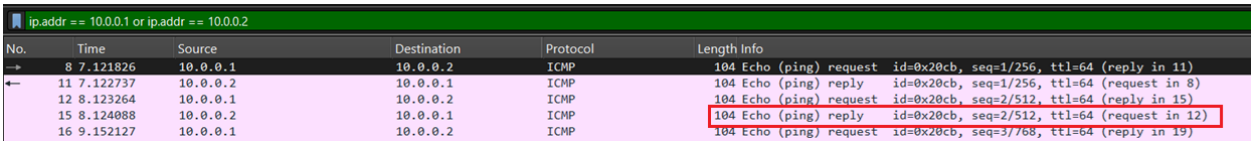


Рис. 5.11. Внутрішній вигляд, знятий на інтерфейсі wg0

9. У звіті пояснити, чому обидва погляди опинилися в одному файлі (`--network container:wg-a` означає спільний мережевий простір імен із wg-a, тобто sniffer технічно перебуває «на кінцевій точці», а не «в мережі між точками»), і чому реальний спостерігач у мережі (без доступу до жодної з кінцевих точок) побачив би виключно перший, зашифрований варіант.

10. Прибрати тестову інфраструктуру після завершення й збереження всіх скріншотів.

```
docker stop wg-a wg-b
docker rm wg-a wg-b
docker network rm wgnet
```

Індивідуальні завдання

Кожен здобувач отримує індивідуальний варіант (табл. 5.2) із власним переліком сайтів для перевірки SNI/ECH, публічним DoT-резолвером для бонусного завдання і адресацією для побудови свого WireGuard-тунелю.

Таблиця 5.2. Варіанти індивідуальних завдань

№	Сайти для аналізу SNI/ECH	Публічний DoT-резолвер	Підмережа для WireGuard-тунелю
1	cloudflare.com research.cloudflare.com	1.1.1.1 (Cloudflare)	10.0.1.0/24
2	discord.com www.discord.com	8.8.8.8 (Google)	10.0.2.0/24
3	youtube.com www.google.com	9.9.9.9 (Quad9)	10.0.3.0/24
4	cloudflareinsights.com www.cloudflarestatus.com	94.140.14.14 (AdGuard DNS)	10.0.4.0/24
5	mozilla.org mozilla-ohttp.fastly.net	149.112.112.112 (Quad9, другорядний)	10.0.5.0/24
6	www.bing.com www.microsoft.com	208.67.222.222 (OpenDNS)	10.0.6.0/24

Завдання 1 (аналіз SNI/ECH). Для кожного із двох сайтів свого варіанта захопити трафік при переході за посиланням, знайти відповідний Client Hello, визначити, чи присутнє розширення encrypted_client_hello, і застосувати практичний критерій з теоретичної частини для визначення, чи це справжній ECH, чи GREASE. Важливо: результат не заданий наперед – підтримка ECH конкретними доменами може змінюватися з часом, тому оцінюється коректність застосованої методики визначення, а не збіг із якоюсь очікуваною відповіддю.

Завдання 2 (DoT). За допомогою контейнера з kdig виконати запит через DoT до свого призначеного резолвера з табл. 5.2, одночасно захопити трафік на порту 853 і підтвердити скріншотом, що це зашифрований TLS-потік, а не звичайний DNS. Приклад команди: `docker run --rm -it instrumentisto/knot-dnsutils sh`, далі `kdig @<резолвер> +tls example.com`.

Завдання 3 (WireGuard). Повторити побудову тунелю з Частини Б, використовуючи власну підмережу з табл. 5.2, і відтворити обидва контрольні захвати (до/після) з відповідними скріншотами.

Контрольні питання

1. Чому шифрування DNS-запитів саме по собі не гарантує повну приватність відвідування сайтів? Яке ще поле TLS-з’єднання видає ім’я сайту?
2. Опишіть структуру TLS Client Hello. Які поля відповідають за узгодження шифронабору, а яке – за ім’я хоста призначення (SNI)?
3. У чому принципова різниця між Outer Client Hello та Inner Client Hello в механізмі ECH?
4. Що таке GREASE ECH і навіщо браузери навмисно додають цей фіктивний елемент до кожного з'єднання, незалежно від підтримки ECH сервером?
5. Сформулюйте практичний критерій, за яким можна відрізнити справжній ECH від GREASE ECH, аналізуючи лише перехоплений трафік (без доступу до приватних ключів).
6. Чому Initial-пакети QUIC можна розшифрувати без жодного приватного ключа сторін з'єднання? Яка специфікація це визначає і з якою метою (яку саме властивість це забезпечує, а яку – ні)?
7. Чим CRYPTO-фрейм у QUIC принципово відрізняється від STREAM-фрейму? Чому спроба «Follow QUIC Stream» на Initial-пакеті не дає результату?
8. Чому традиційні (TCP-орієнтовані) правила глибокої інспекції пакетів (DPI) можуть не спрацьовувати на QUIC-трафіку?
9. Що саме шифрує WireGuard в оригінальному пакеті – лише корисне навантаження чи весь пакет, включно з заголовками? Як це підтверджується структурою захопленого WireGuard Transport Data пакета?
10. Поясніть, чому в межах моделі загроз WireGuard спостерігач, що не має доступу до кінцевої точки тунелю (чи мережі за шлюзом), не може побачити оригінальний (розшифрований) трафік, навіть перехоплюючи весь трафік «на дроті» між кінцевими точками.

11. У сценарії WireGuard типу site-to-site за яких умов зловмиснику не потрібен доступ до самого WireGuard-хоста, щоб побачити трафік у відкритому вигляді?

ЛАБОРАТОРНА РОБОТА 6: Побудова мінімального SOAR-конвеєра: виявлення, оркестрація та автоматичне реагування

Мета: ознайомитися з архітектурою SOAR і самостійно розгорнути мінімальний повний конвеєр «виявлення → оркестрація → автоматичне реагування» на основі відкритих інструментів, простеживши на практиці повний цикл обробки мережевого інциденту.

Теоретична частина

Сучасні центри операцій з кібербезпеки покладаються на комплекс інструментів, де ключові ролі відіграють системи SIEM та SOAR. Платформа SIEM відповідає за збір, кореляцію й зберігання подій безпеки з багатьох джерел, але її результатом, як правило, є лише алерт чи інцидент, що потребує подальшого ручного розгляду аналітиком. Саме тут у гру вступає SOAR, який споживає вже сформовані алерти (від SIEM, систем виявлення вторгнень чи EDR) і виконує заздалегідь визначені, формалізовані сценарії реагування з мінімальною участю людини або зовсім без неї для типових, добре зрозумілих класів інцидентів.

Головна практична цінність SOAR полягає у суттєвому скороченні часу реагування (MTTR) для повторюваних сценаріїв, що дозволяє вивільнити час аналітиків для складних, нетипових розслідувань. Основою автоматизації в SOAR є так званий *playbook* – формалізована послідовність автоматизованих кроків, що запускається за фактом надходження алерту певного типу. Його типова структура складається з трьох логічних частин: прийому й первинної обробки алерту (*ingestion*), збагачення контекстом (наприклад, перевірки репутації IP-адреси через зовнішній сервіс) та, власне, ухвалення рішення з подальшою дією. Остання частина використовує умовну логіку, щоб визначити, чи потрібне реагування, після чого здійснює виклик самої дії. Побудований у цій лабораторній роботі *workflow* платформи n8n, хоч і є мінімальним, буквально відтворює цю структуру: вузол `Webhook` виконує

прийом алерту, вузол `IF` відповідає за ухвалення рішення на основі рівня критичності (*severity*), а вузол `HTTP Request` виконує саму дію.

Для того щоб ініціювати описаний сценарій, потрібне джерело подій, яким у цій роботі виступає Suricata – рушій виявлення вторгнень (IDS/NSM) з відкритим кодом. Ця система генерує структуровані алерти у форматі EVE JSON, який є типовим форматом для споживання реальними SOAR-платформами. Важливо розрізняти типи подій, які Suricata записує у лог-файл, оскільки оркестратор цікавить лише подія типу `alert`. Вона з'являється виключно тоді, коли трафік збігається із завантаженим правилом (сигнатурою), і містить необхідну для SOAR інформацію: назву сигнатури, рівень критичності та IP-адреси джерела й призначення. Інші записи не є приводом для реагування: наприклад, подія `flow` генерується для кожного завершеного мережевого з'єднання (незалежно від того, чи було в ньому щось підозріле) і несе лише метадані сесії, а подія `stats` періодично відображає службову інформацію про сам рушій.

Таблиця 6.1. Типи подій Suricata в eve.json

event_type	**Коли з'являється**	**Значення для SOAR**
`flow`	Для КОЖНОГО завершеного мережевого з'єднання, незалежно від того, чи щось «підозріле» в ньому було	Метадані про сесію (адреси, порти, тривалість); самі по собі не є приводом для реагування
`alert`	Лише тоді, коли трафік збігається із завантаженим правилом (сигнатурою)	Це і є подія, яку споживає SOAR-оркестратор – містить назву сигнатури, severity, IP джерела та призначення
`stats`	Періодично (за замовчуванням раз на кілька секунд роботи рушія)	Службова інформація про сам рушій (лічильники, кількість завантажених правил); може не встигнути записатися при швидкій обробці короткого pcap

Для створення ізольованого та відтворюваного навчального стенда використовується аналіз наперед захопленого pcap-файлу, що запускається за допомогою прапорця `-r`. Такий офлайн-аналіз функціонально еквівалентний живому моніторингу для цілей роботи, оскільки він повністю

детермінований, однаково відтворюється на будь-якому робочому місці та позбавляє потреби у постійному генеруванні трафіку чи наданні прав на захоплення живого мережевого інтерфейсу. Більш того, під час швидкої обробки короткого pcap-файлу події типу `stats` можуть взагалі не встигнути записатися.

Отримавши алерт від Suricata, SOAR має виконати команду реагування, інтегруючись із засобами захисту. Таким чином, суть роботи з SOAR полягає саме в дослідженні принципу оркестрування викликів до різних систем через їхні API: інструмент викликає REST API, щоб дати команду іншій системі виконати дію. Те, що саме приховується за цим API — реальне правило міжмережевого екрана, ізоляція хоста в EDR чи тимчасовий бан на балансувальнику навантаження – для логіки самого SOAR-сценарію не має жодного значення. Розгортання повноцінних продуктів (наприклад, pfSense чи комерційного EDR) лише заради єдиного виклику «заблокувати IP» додало б значний обсяг інфраструктурної складності, не навчаючи нічому додатковому щодо власне SOAR-концепцій. Саме тому замість реального продукту використовується власний мінімальний Flask-застосунок (mock-firewall), який обробляє запит `POST /block {ip}.`, що демонструє той самий принцип інтеграції: у реальному розгортанні той самий workflow міг би звертатися до API справжнього фаєрвола чи EDR без жодної зміни логіки оркестрації, потребуючи лише зміни URL та формату запиту.

Практична частина

Робота виконується на одному Windows-хості з уже встановленим Docker Desktop.

1. Підготовка середовища:

1.1. Створити ізольовану Docker-мережу для всіх компонентів стенду

```
docker network create soarnet
```

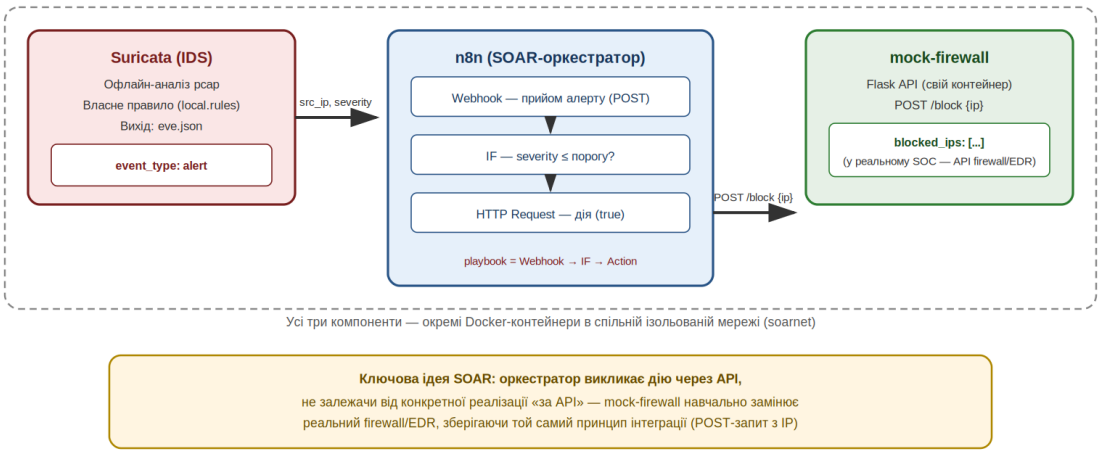


Рис. 6.1. Архітектура мінімального SOAR-конвеєра

2. Генерація підозрілого трафіку для аналізу:

2.1. Підняти умовну «ціль» та тимчасовий контейнер-«sniffer», що ділить її мережевий простір імен (*той самий прийом, що використовувався в лабораторній роботі про безпечні протоколи для перехоплення трафіку всередині Docker-мережі на Windows*).

```
docker run -d --name target --network soarnet nginx:alpine
docker run -d --name soar-sniffer --network container:target
nicolaka/netshoot tcpdump -i any -w /scan.pcap
```

2.2. Виконати сканування портів цілі з окремого контейнера в тій самій мережі – це і буде «підозріла» активність, яку далі має виявити Suricata.

```
docker run --rm --network soarnet nicolaka/netshoot nmap -sS -p
1-1000 target
```

2.3. Зупинити захоплення, скопіювати pcap-файл на хост, прибрати тимчасовий контейнер.

```
docker stop soar-sniffer
docker cp soar-sniffer:/scan.pcap C:\LABs\6\scan.pcap
docker rm soar-sniffer
```

3. Перший запуск Suricata (без власних правил)

3.1 Проаналізувати отриманий pcap-файл офлайн стандартним образом за допомогою Suricata.

```
mkdir C:\LABs\6\output
```

```
docker run --rm -v C:\LABs\6:/pcaps -v C:\LABs\6\output:/output
jasonish/suricata -r /pcaps/scan.pcap -l /output -k none
```

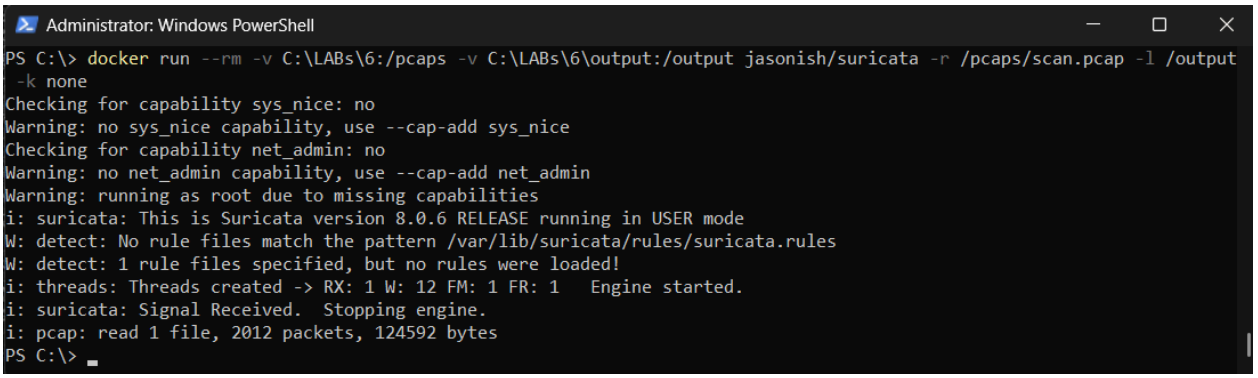


Рис. 6.2. Результат офлайн-аналізу scan.pcap

Важливий практичний момент: образ Suricata «з коробки» не постачається з жодним завантаженим набором правил (типовий шлях /var/lib/suricata/rules/suricata.rules у ньому порожній чи відсутній) – тому в результуючому eve.json на цьому етапі будуть лише записи `event_type: flow` (сирі метадані з'єднань), а полів `event_type: alert` не буде взагалі, незалежно від того, наскільки «підозрілим» був сам трафік. Переконатися в цьому можна командою:

```
type C:\LABs\6\output\eve.json | findstr "event_type"
```

4. Написання власного правила виявлення

4.1. Створити мінімальне, самодостатнє правило виявлення сканування портів за порогом кількості SYN-пакетів від одного джерела за короткий проміжок часу.

Важливо: файл із правилом обов'язково розмістити в ОКРЕМІЙ ТЕЦІ, а не як одиночний файл. Монтування через `docker run -v` одного окремого файлу на Docker Desktop / Windows (WSL2-бекенд) може призвести до того, що в контейнер підключиться порожня тека замість очікуваного файлу – без жодного повідомлення про помилку, просто зі значенням rules_loaded: 0 у статистиці.

```
mkdir C:\LABs\6\rules
```

Створити файл C:\LABs\6\rules\local.rules із вмістом:

```
alert tcp any any -> any any (msg:"SOAR-Lab: TCP SYN scan
detected"; flags:S; threshold: type both, track by_src, count 5,
seconds 3; sid:1000001; rev:1;)
```

5. Повторний запуск Suricata з власним правилом

```
docker run --rm -v C:\LABs\6:/pcaps -v C:\LABs\6\output:/output
-v C:\LABs\6\rules:/etc/suricata/rules jasonish/suricata -r
/pcaps/scan.pcap -l /output -S /etc/suricata/rules/local.rules
-k none
```

Важливо: для короткого одноразового прогону pcap-файлу (обробка триває частки секунди) періодична подія `statistics` може просто не встигнути записатися в eve.json, тому поле `rules_loaded`, яке зазвичай там шукають, може бути відсутнє в принципі, а не дорівнювати нулю. Найнадійнішим джерелом підтвердження є текстовий лог-файл самого рушія:

```
type C:\LABs\6\output\suricata.log
```

У ньому мають з'явитися рядки на кшталт «detect: 1 rule files processed. 1 rules successfully loaded, 0 rules failed, 0 rules skipped» і, наприкінці, «counters: Alerts: 1» – саме ці два рядки є остаточним підтвердженням того, що правило завантажилось і спрацювало (рис. 6.3).

```
[1 - Suricata-Main] 2026-07-22 22:47:31 Notice: threads: Threads created -> RX: 1
W: 12 FM: 1 FR: 1   Engine started.
[16 - RX#01] 2026-07-22 22:47:31 Info: checksum: No packets with invalid
checksum, assuming checksum offloading is NOT used
[16 - RX#01] 2026-07-22 22:47:31 Info: pcap: pcap file /pcaps/scan.pcap end of
file reached (pcap err code 0)
[1 - Suricata-Main] 2026-07-22 22:47:31 Notice: suricata: Signal Received.
Stopping engine.
[1 - Suricata-Main] 2026-07-22 22:47:32 Info: suricata: time elapsed 0.197s
[16 - RX#01] 2026-07-22 22:47:33 Notice: pcap: read 1 file, 2012 packets, 124592
bytes
[1 - Suricata-Main] 2026-07-22 22:47:33 Info: counters: Alerts: 1
```

Рис. 6.3. Підтвердження алерту

6. Встановлення контексту алерту:

```
type C:\LABs\6\output\eve.json | findstr "SOAR-Lab"
```

У знайденому JSON-записі (`"event_type":"alert"`) зафіксувати значення полів `src_ip` і `severity`, які знадобляться на наступних кроках для передачі в SOAR-оркестратор (рис. 6.4).

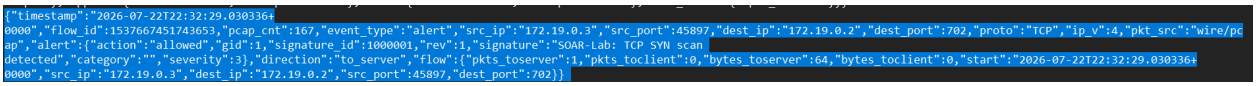


Рис. 6.4. Відновлений контекст алерту: "src_ip":"172.19.0.3", "severity":3

7. Розгортання оркестратора n8n

```
docker run -d --name n8n --network soarnet -p 5678:5678
n8nio/n8n
```

Відкрити в браузері http://localhost:5678 і створити локальний обліковий запис власника (усе відбувається офлайн, без зовнішньої реєстрації).

8. Побудова mock-firewall:

8.1. Створити мінімальний Flask-застосунок, що імітує API засобу реагування

```
mkdir C:\LABs\6\mock-firewall
```

Створити файл app.py:

```
from flask import Flask, request
app = Flask(__name__)
blocked = []

@app.route('/block', methods=['POST'])
def block():
    ip = request.json.get('ip')
    blocked.append(ip)
    print(f'BLOCKED: {ip}', flush=True)
    return {'status': 'ok', 'blocked_ips': blocked}, 200

app.run(host='0.0.0.0', port=8080)
```

Створити Dockerfile:

```
FROM python:3.12-slim
RUN pip install flask
COPY app.py .
CMD ["python", "app.py"]
```

8.2. Зібрати образ файрволу і запустити контейнер у тій самій мережі:

```
docker build -t mock-firewall .
docker run -d --name mock-firewall --network soarnet -p
8080:8080 mock-firewall
```

9. Перевірка mock-firewall окремо

```
Invoke-RestMethod -Uri "http://localhost:8080/block" -Method
Post -ContentType "application/json" -Body '{"ip": "1.2.3.4"}'
```

Очікувана відповідь: `{"status":"ok","blocked_ips":["1.2.3.4"]}`.

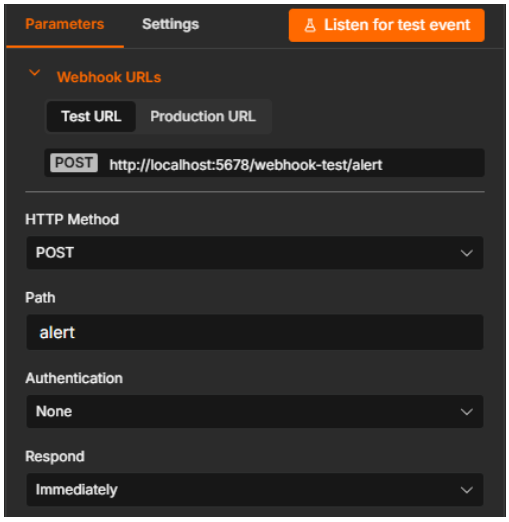
10. Побудова Workflow (*playbook*) у n8n:

10.1. Створити новий Workflow з трьох вузлів

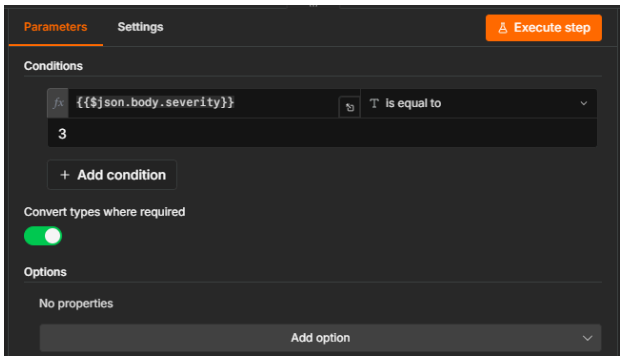
a) `Webhook` – Method: POST, Path: alert (рис. 6.5a);

b) `IF` – умова `{{$json.body.severity}} is equal to 3` (порівняння з тим значенням severity, яке присвоює правило Suricata) (рис. 6.5b);

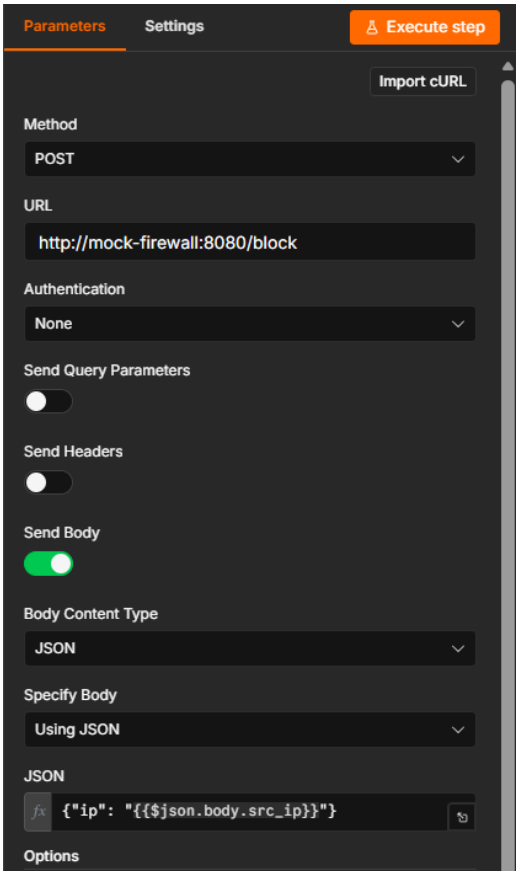
c) `HTTP Request` (підключений до виходу `true` вузла `IF`) – Method: POST, URL: http://mock-firewall:8080/block, Send Body: увімкнене, Body Content Type: JSON, тіло: {"ip": "{{$json.body.src_ip}}"} (рис. 6.5c).



a



b



c

Рис. 6.5. Налаштування вузлів: `Webhook` (a), `IF` (b), `HTTP Request` (c)

10.2. Протестувати вузол `IF` окремо (*execute step*). Якщо вузол видасть помилку на кшталт «'3' is a number but was expecting a string» – це типова невідповідність типів low-code-платформи: вхідне значення severity приходить як число, а умова порівнюється як рядок. Виправляється увімкненням перемикача «Convert types where required» у налаштуваннях вузла `IF` (або зміною типу порівнюваного значення з String на Number).

10.3. Надіслати тестовий алерт на тестовий шлях `Webhook` (діє лише під час активного редагування Workflow, доки він не активований).

```
Invoke-RestMethod -Uri
"http://localhost:5678/webhook-test/alert" -Method Post
-ContentType "application/json" -Body '{"src_ip": "172.19.0.3",
"severity": 3, "signature": "SOAR-Lab: TCP SYN scan detected"}'
```

Після успішного тестового виконання всі три вузли мають «позеленіти» без позначок помилки; вузол `HTTP Request` у своєму виводі (OUTPUT) повинен показати відповідь mock-firewall.

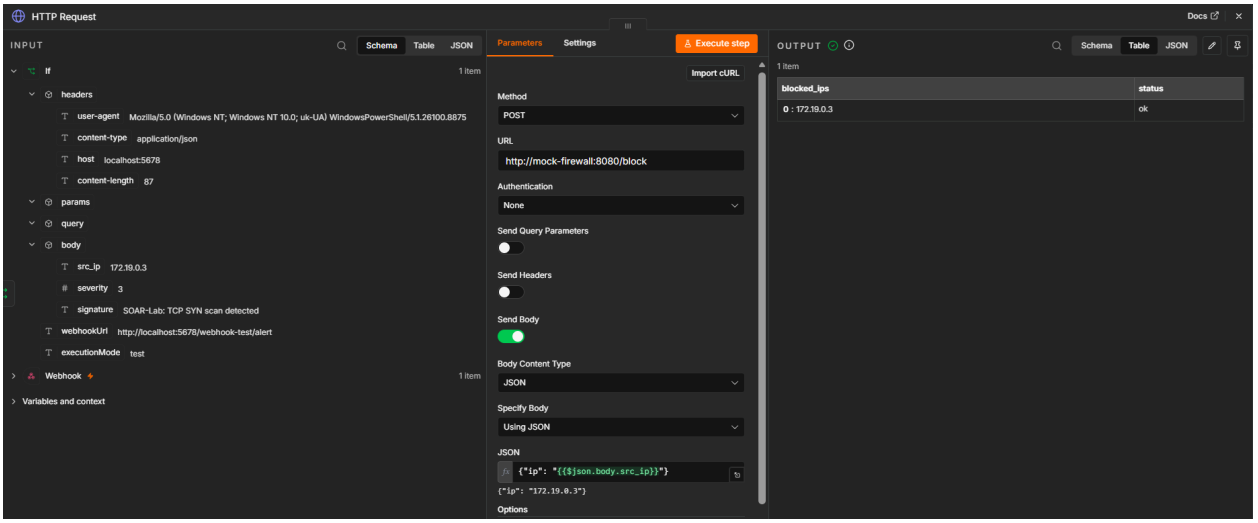


Рис. 6.6. Відповідь mock-firewall

11. Активація Workflow і перевірка «бойового» виклику:

11.1. Увімкнути перемикач Active workflow.

11.2. Надіслати той самий алерт уже на постійний шлях, підставивши реальний src_ip, знайдений в п. 6.

```
Invoke-RestMethod -Uri "http://localhost:5678/webhook/alert"
-Method Post -ContentType "application/json" -Body '{"src_ip":
"172.19.0.3", "severity": 3, "signature": "SOAR-Lab: TCP SYN
scan detected"}'
```

11.3. Перевірити результат безпосередньо в контейнері засобу реагування.

```
docker logs mock-firewall
```

Важливо: IP-адреса джерела запиту, яку побачить mock-firewall, відрізняється між тестовим і активним викликом. При тестовому виклику запит логічно проходить через хост (адреса шлюзу Docker-мережі), тоді як після активації Workflow n8n сам виконує HTTP-виклик зсередини свого контейнера в межах мережі soarnet, тобто адресою джерела буде вже реальна внутрішня IP-адреса контейнера n8n (рис. . Це наочно ілюструє відмінність контексту виконання між режимом редагування/тестування та активним, «бойовим» станом workflow.

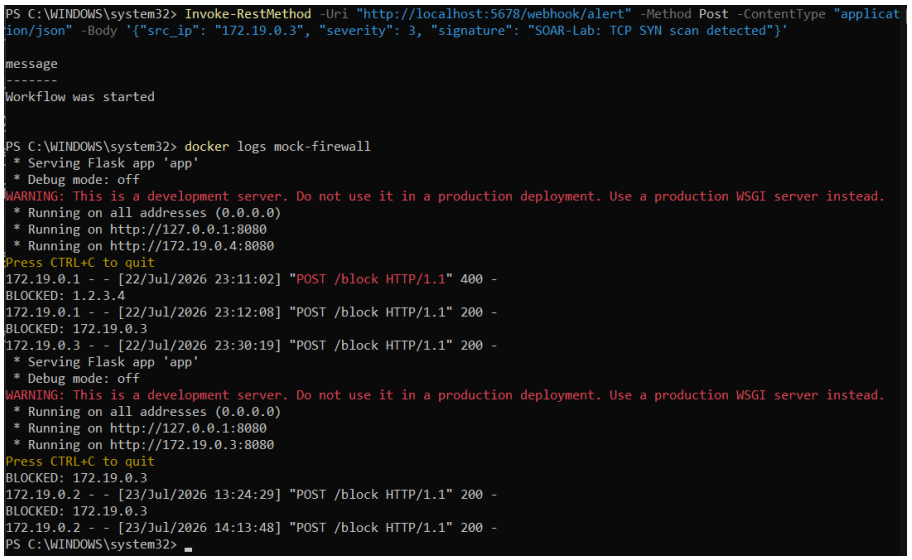


Рис. 6.7. Лог mock-firewall

12. Завершення роботи:

Прибрати тестову інфраструктуру

```
docker stop target n8n mock-firewall
docker rm target n8n mock-firewall
docker network rm soarnet
```

Індивідуальні завдання

Кожен здобувач отримує індивідуальний сценарій виявлення (табл. 6.2). Завдання виконується самостійно за тією самою логікою, що й базовий приклад:

— згенерувати відповідний тип трафіку в ізольованій Docker-мережі;

— самостійно скласти правило Suricata під свій сценарій (спираючись на підказку в таблиці й документацію Suricata щодо синтаксису правил);

— підтвердити спрацювання правила через suricata.log та eve.json;

— підключити той самий (або новостворений) n8n-workflow і mock-firewall для автоматичного реагування на цей алерт;

— оформити звіт зі скріншотами кожного етапу.

Для варіантів, що потребують додаткової цілі, відмінної від звичайного веб-сервера (наприклад, варіант 8 із FTP), дозволяється підняти додатковий контейнер відповідного призначення в тій самій мережі soarnet.

Таблиця 6.2. Варіанти індивідуальних завдань

№	Сценарій виявлення	Підказка щодо власного правила Suricata	Генерація трафіку
1	TCP SYN-скан (*інший поріг спрацювання*)	flags:S; threshold: type both, track by_src, count 10, seconds 5	nmap -sS -p 1-1000 <*ціль*>
2	ICMP-флуд (*підозріла кількість ping за короткий час*)	заголовок правила: alert icmp ... (proto = icmp замість tcp); threshold: type both, track by_src, count 20, seconds 2	цикл з кількох десятків ping поспіль з одного контейнера до цілі
3	UDP-скан портів	заголовок правила: alert udp ... (proto = udp; *опція flags тут не застосовна, бо в UDP немає прапорців TCP*); threshold: type both, track by_src, count 10, seconds 5	nmap -sU -p 1-200 <*ціль*>
4	Підозрілий User-Agent (*сигнатура сканера вразливостей*)	http.user_agent; content:"sqlmap"; nocase	curl -A "sqlmap/1.7" http://<*ціль*>/
5	Повторні звернення до чутливого шляху (*можлива спроба експлуатації*)	http.uri; content:"/wp-admin/admin-ajax.php"; threshold: track by_src, count 5, seconds 10	curl http://<*ціль*>/wp-admin/admin-ajax.php *кілька разів поспіль*

6	Численні розірвані TCP-з'єднання (*ознака сканування чи недоступних портів*)	flags:R; threshold: type both, track by_src, count 10, seconds 5	звернення nmap/curl до діапазону портів, де більшість закриті
7	Підозріло великий обсяг переданих даних (*можлива ексфільтрація*)	threshold за кількістю великих пакетів від джерела (track by_src) або dsize за порогом	docker exec у контейнер-джерело: cat /dev/urandom | curl -T - http://<ціль>/upload (*чи аналогічна передача великого файлу*)
8	Передача облікових даних у відкритому FTP	content:"PASS "; nocase (*у FTP-трафіку, порт 21*)	підняти контейнер fauria/vsftpd як ціль; звернутися curl ftp://user:pass@<*ціль*>/
9	Порт-скан через detection_filter (*альтернатива threshold*)	detection_filter: track by_src, count 15, seconds 5	nmap -sS -p 1-2000 <*ціль*>
10	DNS-запит до домену з підозрілим TLD	dns.query; content:".xyz"; nocase (*або інший TLD за варіантом*)	docker exec <*контейнер*> nslookup test.xyz (*або curl до такого домену*)

Контрольні питання

1. Що таке SOAR і чим ця концепція принципово відрізняється від SIEM? Поясніть, який компонент лабораторного стенду відповідає кожній із двох концепцій.
2. Що таке playbook у термінології SOAR?
3. Чому в eve.json Suricata з'являються записи з event_type: flow навіть тоді, коли жодне правило не спрацювало? Чим це відрізняється від event_type: alert?
4. Чому образ Suricata «з коробки» не генерує жодних алертів без додаткового кроку? Що конкретно довелося зробити, щоб правило запрацювало?
5. У чому різниця між тестовим (webhook-test) і активним (webhook) шляхами виклику `Webhook`-вузла в n8n? Чому активація Workflow є обов'язковим кроком перед «бойовим» використанням?

6. Чому виклик `HTTP Request` із гілки `IF` повернув різні джерельні IP-адреси при тестовому та активному запуску (з боку mock-firewall)? Що це говорить про відмінність контексту виконання?
7. Навіщо в цій роботі використано власний мінімальний mock-firewall API замість розгортання повноцінного продукту (pfSense, EDR тощо)? Що б довелося змінити в n8n-workflow, якби замість mock-firewall підключався реальний продукт?
8. Що таке «збагачення» (*enrichment*) в контексті SOAR-плейбука, і яким вузлом його можна було б реалізувати в нашому Workflow (навіть якщо ми цього не робили практично)?
9. Чому важливо, щоб дія (виклик `HTTP Request`) виконувалася лише за певної умови (`IF`), а не безумовно для кожного вхідного алерту?

ЛАБОРАТОРНА РОБОТА 7: Емуляція супротивника та моделювання атак: MITRE Caldera й Atomic Red Team

Мета: ознайомитися з методами контрольованої емуляції дій супротивника шляхом відтворення технік матриці MITRE ATT&CK за допомогою двох принципово різних інструментів – платформи керованої C2-емуляції MITRE Caldera та автономного фреймворку атомарних тестів Atomic Red Team, – і на основі практичного досвіду сформулювати обґрунтовані рекомендації щодо вибору інструменту залежно від задачі.

Теоретична частина

У сучасних реаліях ландшафт кіберзагроз розвивається експоненціально, змушуючи організації переходити від реактивної моделі захисту («гасіння пожеж» після інциденту) до проактивної. Традиційні методи, такі як класичне сканування вразливостей або періодичний пентест, вже не дають повної картини захищеності. Вони фіксують лише статичний стан системи, але не показують, як саме зловмисник буде просуватися всередині інфраструктури, які техніки обходу засобів захисту він використає та чи зможуть наявні системи моніторингу (SIEM/EDR) вчасно виявити цю активність.

Саме тому концепція емуляції супротивника (*Adversary Emulation*) стала фундаментальним елементом сучасної практичної кібербезпеки. На відміну від хаотичних атак, емуляція є чітко структурованим, легітимним та контрольованим процесом відтворення реальної поведінки хакерських угруповань. Проведення таких симуляцій у спеціально виділеному, ізольованому середовищі дозволяє безпечно випробовувати міцність оборони, не наражаючи на ризик критичні бізнес-процеси та реальні дані організації.

Головним орієнтиром та «словником» для такої діяльності є глобальна база знань MITRE ATT&CK, яка дає змогу аналітику систематизувати тисячі реальних тактик, технік та процедур (TTPs), які використовують зловмисники

на різних етапах атаки. Емуляція у контрольованому середовищі дає змогу не просто теоретично вивчати цю матрицю, а практично перевірити, які саме клітинки MITRE ATT&CK покриті наявними системами детектування, а які залишаються «сліпими зонами».

Дана лабораторна робота покликана сформувати практичні навички роботи з передовим інструментарієм для тестування систем захисту. Замість ручного відтворення кожної команди, сучасна індустрія використовує автоматизовані фреймворки, що дозволяють будувати складні сценарії атак або швидко тестувати конкретні вектори. Основними інструментами в рамках цієї роботи виступають платформи Caldera та Atomic Red Team.

MITRE Caldera – це потужна кібербезпекова платформа з відкритим вихідним кодом, створена безпосередньо дослідниками інституту MITRE на базі матриці ATT&CK. Головна її мета – повна автоматизація дій супротивника та рутинних операцій з тестування безпеки.

Caldera побудована на основі агентної архітектури. Спеціальні легковагові агенти (наприклад, Sandcat) встановлюються на цільові хости в тестовій мережі та очікують вказівок від центрального сервера керування (C2-сервера).

Ключові особливості фреймворку Caldera:

— Автономні операції. Завдяки логічним алгоритмам (*Planners*), платформа здатна самостійно приймати рішення в процесі виконання атаки: якщо якась техніка збору інформації чи закріплення в системі зазнає невдачі, Caldera автоматично коригує свій шлях і обирає альтернативний варіант дій.

— Сценарні профілі (*Adversary Profiles*), які дозволяють об’єднувати десятки окремих технік у комплексні ланцюжки атак, що імітують поведінку конкретних відомих APT-груп (наприклад, APT29 або FIN6).

— Гнучка розширюваність. Завдяки плагінній системі, фреймворк легко інтегрується з іншими інструментами захисту, підтримує модулі для збору аналітики та візуалізації операцій в реальному часі.

Atomic Red Team (ART) – це відкрита бібліотека простих, чітко сфокусованих та детально описаних тестів, розроблена компанією Red Canary. Кожен такий тест (званий «атомом») безпосередньо прив'язаний до конкретної техніки чи підтехніки в матриці MITRE ATT&CK.

Філософія Atomic Red Team полягає у простоті, швидкості та точності тестування. Замість розгортання складної інфраструктури, ART дозволяє безпечно виконати одну або кілька команд на хості, щоб миттєво перевірити реакцію засобів захисту.

Ключові переваги підходу Atomic Red Team:

— Атомарність. Кожен тест перевіряє лише один вузький аспект поведінки супротивника (наприклад, додавання конкретного ключа в реєстр Windows, запуск замаскованого PowerShell-скрипта або спробу дампу пам'яті процесу lsass.exe).

— Зручність автоматизації. Всі тести описані в уніфікованому та зрозумілому форматі YAML, що дозволяє легко запускати їх як вручну, так і автоматизовано за допомогою спеціального модуля керування Invoke-AtomicRedTeam.

— Миттєвий зворотний зв'язок (*Feedback Loop*). Інженери з кібербезпеки можуть запустити конкретний «атом» і одразу перевірити логи в SIEM або сповіщення в EDR-системі, що дозволяє оперативно валідувати правила детектування та закривати діри в обороні без проведення масштабних і дорогих кібернавчань.

Порівняльний аналіз MITRE Caldera та Atomic Red Team демонструє, що попри спільну фундаментальну базу (матрицю MITRE ATT&CK), ці інструменти мають принципово різну філософію, архітектуру та сфери застосування (табл. 7.1).

Таблиця 7.1. Порівняльний аналіз MITRE Caldera та Atomic Red Team

Характеристика	**MITRE Caldera**	**Atomic Red Team**
Тип інструменту	Комплексна C2-платформа для автоматизованої оркестрації атак.	Дискретна бібліотека тестових скриптів («атомів») для перевірки конкретних технік.

Архітектура	Клієнт-серверна: центральний C2-сервер керує агентами, встановленими на цільових хостах.	Локальна: тести запускаються безпосередньо на хості через PowerShell-модуль Invoke-AtomicRedTeam.
Складність розгортання	*Середня* / *Висока*: вимагає підготовки сервера, налаштування мережевих допусків та встановлення агентів.	*Низька*: достатньо імпортувати модуль та завантажити YAML-файли з тестами.
Рівень автономності	*Високий*: алгоритми самостійно адаптують сценарій атаки залежно від реакції системи.	*Низький*: кожен тест запускається оператором вручну або за чітко прописаним лінійним скриптом.
Характер атаки	*Динамічний ланцюжок*: імітує зв'язану послідовність дій зловмисника в часі та просторі (рух по мережі).	Ізольований укол: перевіряє одну конкретну техніку на одному хості «тут і зараз».
Головний споживач	*Purple Teams*, досвідчені *Red Teams*, зрілі SOC-центри для комплексних кібернавчань.	*Blue Teams*, SOC-аналітики, системні адміністратори для швидкої валідації систем захисту.

Таким чином, вибір між Caldera та Atomic Red Team повністю визначається поточними задачами, зрілістю процесів безпеки та наявним часом.

Оптимальні сценарії для використання MITRE Caldera:

— Комплексна емуляція APT-групи (*End-to-End Emulation*): Caldera дозволяє запустити повний профіль атаки: від початкового закріплення до латерального переміщення (*Lateral Movement*) мережею та ексфільтрації даних.

— Тестування реакції Blue Team в реальному часі: Caldera діє автономно та «тихо», що дозволяє перевірити не просто софт (SIEM/EDR), а швидкість та адекватність реакції живих аналітиків SOC на складну, багатовекторну атаку, що розвивається динамічно.

— Оцінка ефективності захисту в динаміці: оскільки Caldera вміє обходити перешкоди (якщо одна техніка заблокована антивірусом, вона спробує іншу), з’являється можливість отримати реалістичну картину того, на якому саме етапі зловмисник все ж таки зможе пробити оборону.

Оптимальні сценарії для використання Atomic Red Team:

— Швидка валідація нових SIEM/EDR-правил.

— Обмежений час та ресурси: у випадку відсутності можливості розгортання серверів та складної інфраструктури при необхідності негайної перевірки захищеності хоста.

— Створення базових регресійних тестів безпеки: ART ідеально інтегрується в CI/CD-пайплайни або регулярні таски Windows Task Scheduler / Cron для перевірки, чи не «злетіли» налаштування безпеки на серверах після чергового оновлення ОС.

Практична частина

Частина А. MITRE Caldera

1. Розгортання стенду (рис. 7.1).

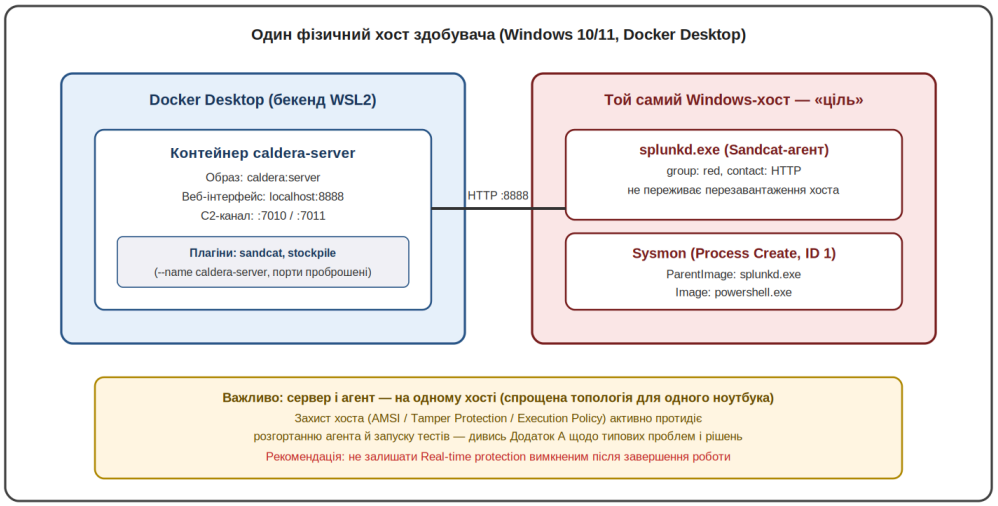


Рис. 7.1. Архітектура стенду для розгортання MITRE Caldera

1.1. Встановити Docker Desktop для Windows (версія AMD64 – стандартна 64-бітна архітектура, підходить для процесорів Intel і AMD), під час процесу інсталяції залишити позначку «Use WSL 2 instead of Hyper-V». Перевірити працездатність запуском тестового образу:

```
docker run hello-world
```

1.2. Встановити Git for Windows, клонувати репозиторій Caldera[4] з явною забороною підміни кінців рядків (`core.autocrlf=false`):

[4] На час виконання роботи поточна гілка фреймворку може відрізнятися від протестованої версії 5.1.0.

```
git clone -c core.autocrlf=false
https://github.com/mitre/caldera.git --recursive --branch 5.1.0
cd caldera
copy conf\default.yml conf\local.yml
```

1.3. Зібрати образ і запустити контейнер з явним іменем.

```
docker build --build-arg WIN_BUILD=true . -t caldera:server
docker run -d -p 7010:7010 -p 7011:7011/udp -p 7012:7012 -p
8888:8888 --name caldera-server caldera:server
```

1.4. Перевірити журнал зборки

```
docker logs caldera-server
```

Якщо наприкінці лог-файлу є рядок «All systems ready.», спробувати відкрити http://localhost:8888, увійти в адміністративну панель (логін: *red*, пароль: *admin*) (рис. 7.2), одразу змінити пароль.

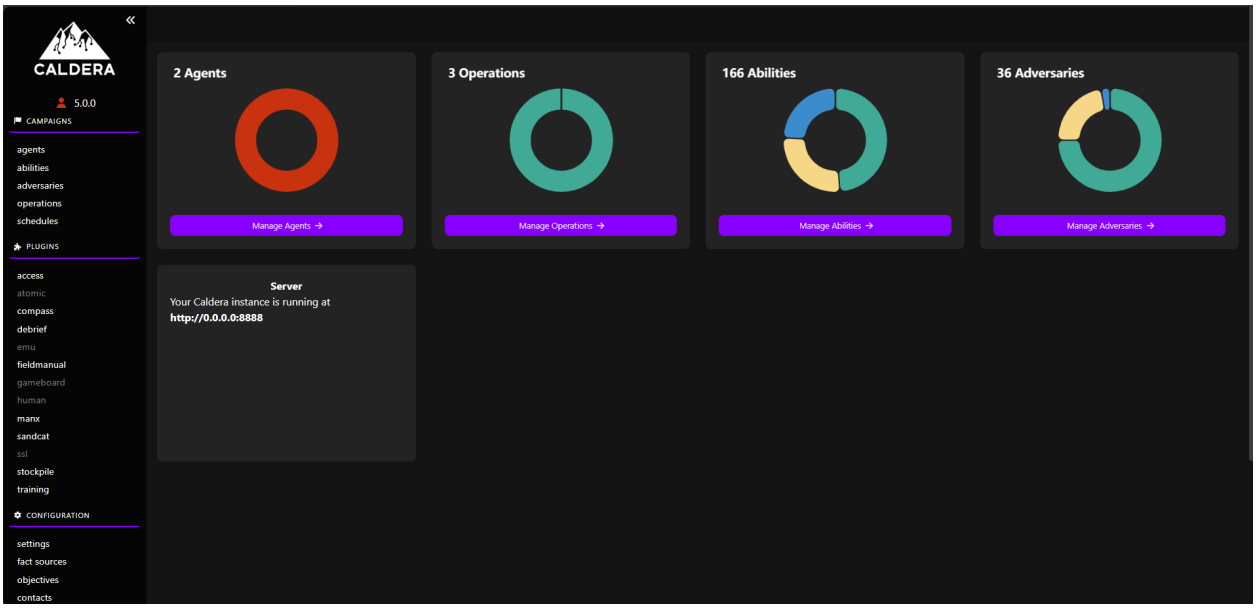


Рис. 7.2. Вікно відкритої адміністративої панелі MITRE Caldera

1.5. Розгорнути агент Sandcat (Agents → Deploy an agent → Windows → HTTP) (рис. 7.2). Перед виконанням згенерованої команди в PowerShell (від імені адміністратора) обов'язково[5]:

```
Set-MpPreference -DisableRealtimeMonitoring $true
```

[5] Виконувати такі дії на особистому ноутбуці без ізоляції – дуже ризикована практика. Після виконання роботи слід повернути real-time захист: Set-MpPreference -DisableRealtimeMonitoring $false

(Windows Defender / AMSI блокує команду розгортання як «*malicious content*» – це очікувано; якщо не допомагає, причина в Tamper Protection, див. розділ «Можливі складнощі»). Перевірити у вкладці Agents статус розгорнутого агенту, він має бути «alive, trusted» (рис. 7.3).

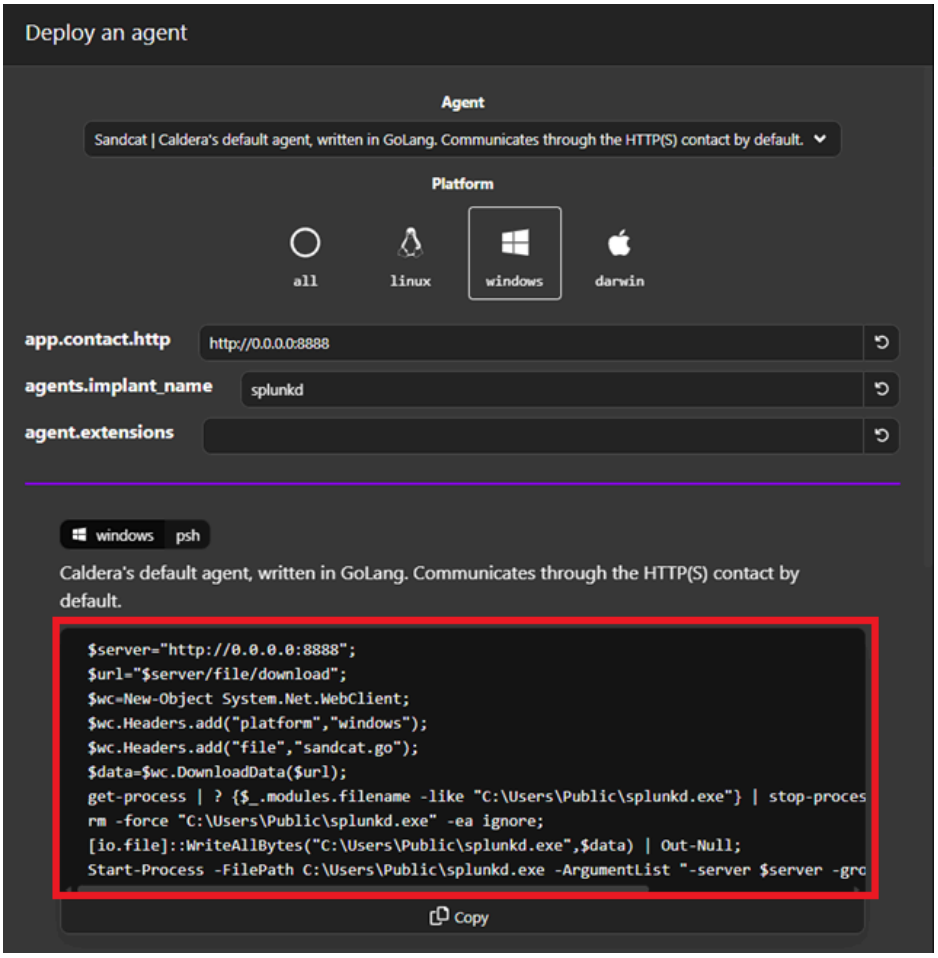


Рис. 7.2. Розгортання Sandcat (червоним прямокутником виділена команда, якою Sandcat має бути встановлений через PowerShell)

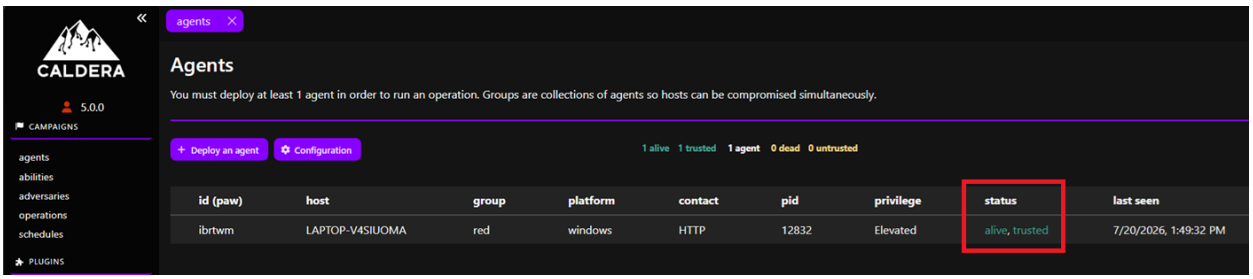


Рис. 7.3. Працюючий Sandcat

2. Створення профілю супротивника.

2.1. У вкладці Adversaries створити профіль (наприклад, Discovery-Baseline) (рис. 7.4), додати три здатності (*abilities*) з табл. 7.2[6].

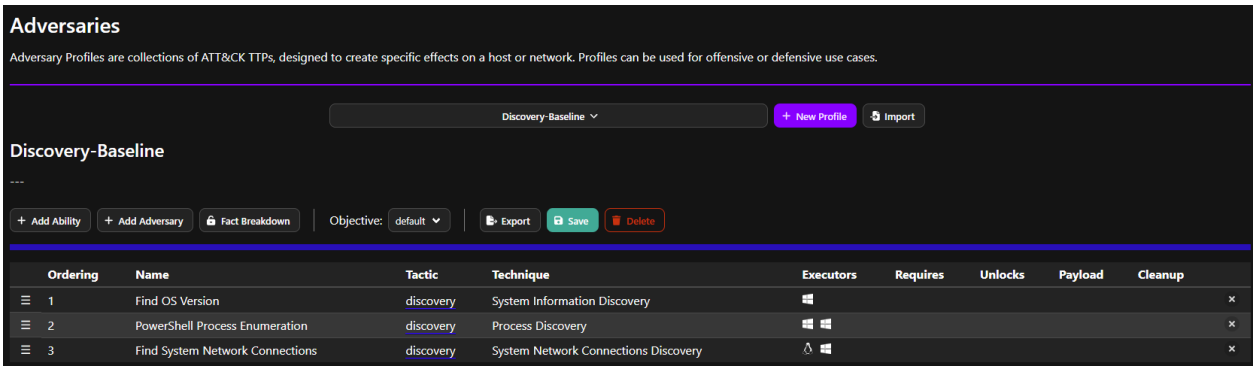


Рис. 7.4. Створений профіль

Таблиця 7.2. Здатності створеного профілю Discovery-Baseline

ATT&CK ID	**Ability**	**Фактична команда**	**Тактика**
T1082	Find OS Version	[environment]::OSVersion.Version	discovery
T1057	PowerShell Process Enumeration	get-process >> $env:APPDATA\vmtools.log; cat $env:APPDATA\vmtools.log	discovery
T1046	Find System Network Connections	netstat -anto; Get-NetTCPConnection	discovery

3. Побудова і проведення операції

3.1. У вкладці Operations створити операцію (рис. 7.5) і запустити її.

[6] Бажано попередньо перевірити команди кожної здатності на відсутність запису у зовнішній файл без виводу, залежності від Facts (замок у стовпці Requires) чи зовнішніх payload'ів (дзвіночок у стовпці Payload).

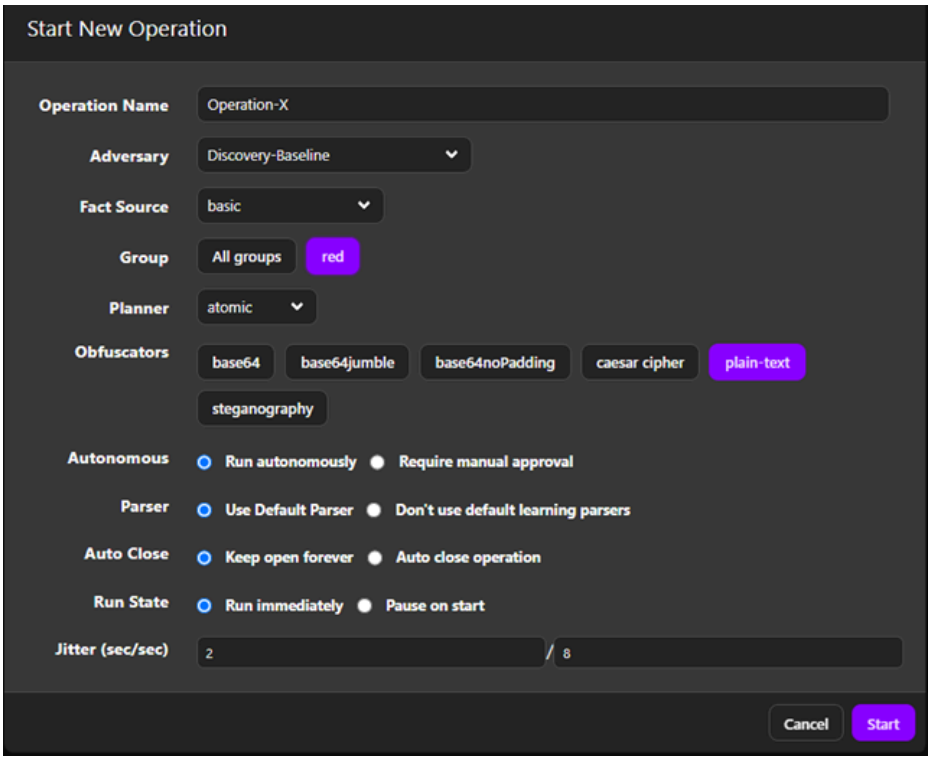


Рис. 7.5. Процес створення операції

3.2. Переконатися, що всі три здатності отримали статус success і мають непорожній Link Output (рис. 7.6).

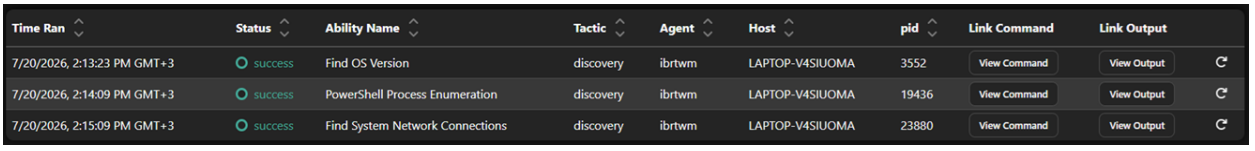


Рис. 7.6. Відтворення операції

4. Аналіз артефактів операції.

4.1 Артефакти, які можна дослідити у MITRE Caldera:

— Find OS Version (рис. 7.7). Аналітичний висновок для атакуючого: Поєднання параметрів Major: 10, Minor: 0 та Build: 26200 чітко ідентифікує цільову операційну систему як Windows 11 (сучасні збірки гілки розробки/Canary або новітні корпоративні релізи). Для зловмисника цей вивід є критично важливим: він підтверджує архітектуру системи та дозволяє відсікти завідомо недієві або застарілі

експлойти, що були розраховані на Windows 7/8 чи старіші білди Windows 10.

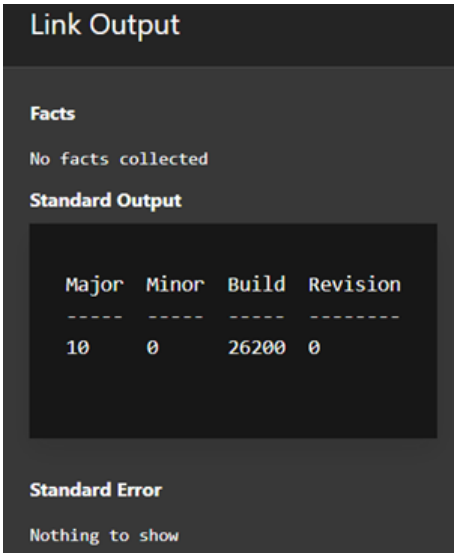


Рис. 7.7. Артефакт здатності Find OS Version

— PowerShell Process Enumeration (рис. 7.8). Вивід представлений у вигляді класичної системної таблиці оболонки PowerShell, що генерується командлетом Get-Process (або його аліасом gps). Важливо розуміти значення метрик, які операційна система повертає атакуючому:

Handles: Кількість відкритих дескрипторів об’єктів (файли, ключі реєстру, потоки), які використовує процес.
NPM(K) (Non-Paged Memory): Обсяг несторінкової пам’яті в кілобайтах, яка не може бути скинута у файл підкачки на жорсткий диск (критично важлива пам’ять ядра).
PM(K) (Paged Memory): Обсяг сторінкової пам’яті в кілобайтах, яка може скидатися у віртуальну пам’яті.
WS(K) (Working Set): Обсяг фізичної оперативної пам’яті, що виділений процесу безпосередньо в цей момент.
CPU(s): Час процесора в секундах, витрачений на обробку цього процесу з моменту його запуску.
Id (PID): Унікальний ідентифікатор процесу в системі.
SI (Session ID): Ідентифікатор сесії. Процеси із SI = 0 (наприклад, AggregatorHost) працюють у системній сесії (сервіси, ізольовані від користувача), а процеси із SI = 1 працюють в інтерактивній сесії поточного користувача.
ProcessName: Ім’я виконуваного файлу без розширення .exe.

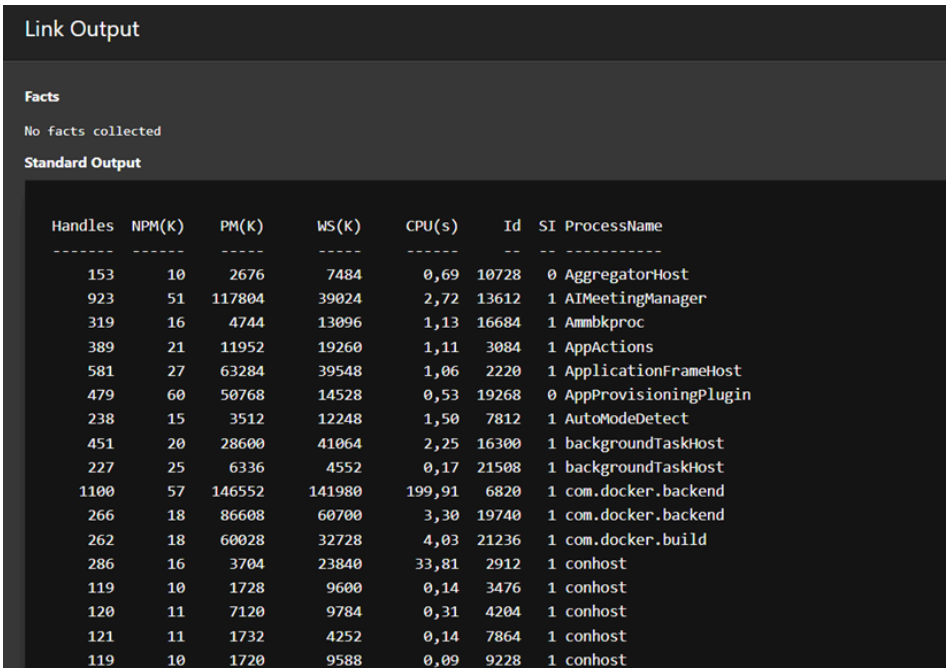


Рис. 7.8. Артефакт здатності PowerShell Process Enumeration

— Find System Network Connections (рис. 7.9). Зловмисник отримав список IP-адрес, пов’язаних із цільовою системою (це можуть бути адреси локальних мережевих інтерфейсів, віртуальних адаптерів, шлюзів або активних мережевих з’єднань).

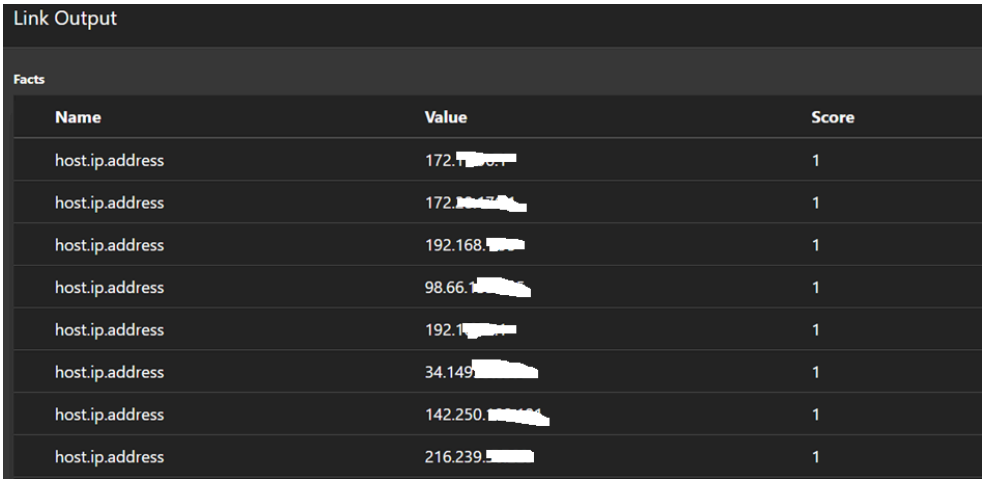


Рис. 7.9. Артефакт здатності Find System Network Connections

4.2. Дослідження артефактів операції на цільовому хості за допомогою Sysmon (легітимний інструмент Sysinternals).

4.2.1. Встановити Sysmon

```
Invoke-WebRequest -Uri
"https://download.sysinternals.com/files/Sysmon.zip" -OutFile
"$env:TEMP\Sysmon.zip"
Expand-Archive "$env:TEMP\Sysmon.zip" -DestinationPath
"$env:TEMP\Sysmon" -Force; cd "$env:TEMP\Sysmon"; .\Sysmon64.exe
-accepteula -i
```

4.2.2. Повторно запустити операцію в Caldera, щоб Sysmon встиг зафіксувати дочірні процеси.

4.2.3. Запустити Event Viewer (eventvwr.msc), перейти в Applications and Services Logs → Microsoft → Windows → Sysmon → Operational. Обрати фільтр «Event ID = 1». Для кожної здатності знайти подію з наступними параметрами:

ParentImage = splunkd.exe,

Image = powershell.exe,

CommandLine відповідає табл. 7.2.

Звірити час події з Time Ran у Caldera та зберегти скріншоти (рис. 7.10).

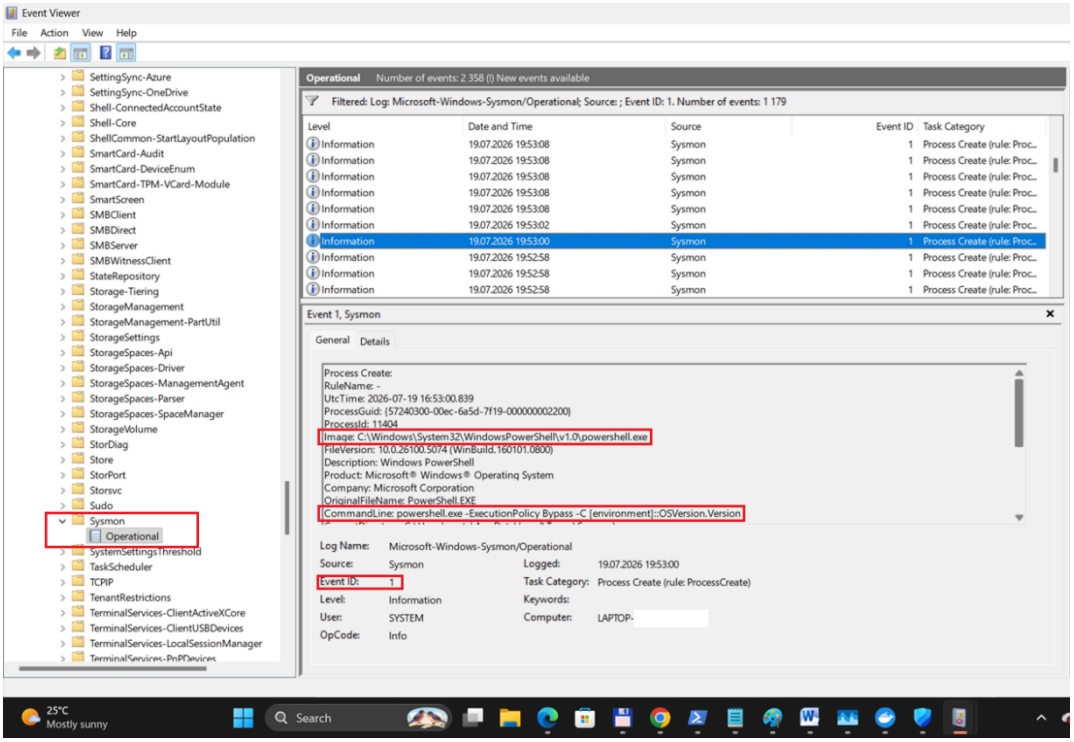


Рис. 7.10. Приклад події що відповідає здатності Find OS Version (T1082)

Частина Б. Atomic Red Team

Ця частина виконується всіма здобувачами однаково (на відміну від Caldera, де індивідуальні варіанти наведено окремо в розділі «Індивідуальні завдання») – мета цієї частини не в ознайомленні з різноманіттю технік, а в порівнянні того, як один і той самий інструментарій ATT&CK поводиться в іншій архітектурі виконання.

1. Встановлення ART.

1.1. У PowerShell від імені адміністратора підготувати сесію та встановити модуль.

```
Set-ExecutionPolicy Bypass -Scope Process -Force
Set-MpPreference -DisableRealtimeMonitoring $true
IEX                                                    (IWR
'https://raw.githubusercontent.com/redcanaryco/invoke-atomicredt
eam/master/install-atomicredteam.ps1'          -UseBasicParsing);
Install-AtomicRedTeam -getAtomics -Force
```

2. Проведення тих самих технік.

2.1. Для кожної техніки необхідно перед застосуванням провести аналіз можливих варіантів її відтворення, перевірити передумови і спосіб виконання і обрати для початку найпростіший варіант відтворення (без зовнішніх інструментів на кшталт WinPEAS / PowerSharpPack / Nmap). Обрані тести наведено в табл. 7.3.

Таблиця 7.3. Обрані техніки для демонстрації

ATT&CK ID	**Atomic Test**	**Команда (скорочено)**	**Кількість процесів**
T1082	Test #1	systeminfo & reg query ...; hostname.exe; whoami.exe	3
T1057	Test #3	Get-Process	1
T1046	Test #10	Port-Scanning /24 Subnet with PowerShell	1

2.2. Відтворити техніки:

```
Invoke-AtomicTest T1082 -TestNumbers 1 -GetPrereqs
Invoke-AtomicTest T1082 -TestNumbers 1
Invoke-AtomicTest T1057 -TestNumbers 3 -GetPrereqs
```

```
Invoke-AtomicTest T1057 -TestNumbers 3
Invoke-AtomicTest T1046 -TestNumbers 10 -GetPrereqs
Invoke-AtomicTest T1046 -TestNumbers 10
```

3. Аналіз артефактів тестування.

3.1. У тому самому фільтрі Sysmon (Event ID 1) знайти нові події – цього разу ParentImage вказуватиме на powershell.exe (сесію, у якій виконувався Invoke-AtomicTest), а не на splunkd.exe, оскільки Atomic Red Team не має проміжного агента-посередника. Зберегти скріншоти для кожної техніки.

4. Виконати очищення там, де воно передбачене.

```
Invoke-AtomicTest T1057 -TestNumbers 3 -Cleanup
Invoke-AtomicTest T1046 -TestNumbers 10 -Cleanup
```

Частина В. Порівняння результатів роботи двох фреймворків

Технічно найцікавіший матеріал для порівняння дає саме T1082 (*System Information Discovery*), оскільки на цій техніці найяскравіше видно принципову різницю в тому, як кожен інструмент трактує поняття «одна техніка».

Caldera («Find OS Version»): одна здатність – один процес, одна команда `[environment]::OSVersion.Version`, один рядок Process Create в Sysmon. Результат тестування цієї техніки є лише чотири числа (Major / Minor / Build / Revision), тобто мінімальний, точковий обсяг даних (табл. 7.2, рис. 7.7).

Atomic Red Team (T1082, Test #1): один «атомарний тест» фактично розгортається у ТРИ окремі процеси з одним батьківським PID = 20832 (сесія PowerShell, у якій виконано Invoke-AtomicTest): (1) cmd.exe з командою `systeminfo & reg query HKLM\SYSTEM\...\Disk\Enum` (рис. 7.11), (2) окремий виклик hostname.exe (рис. 7.12), (3) окремий виклик whoami.exe (рис. 7.13). Обсяг зібраних даних значно більший – повний звіт про «залізо», версію ОС, мережеві карти, встановлені оновлення, стан VBS / Secure Boot, домен тощо.

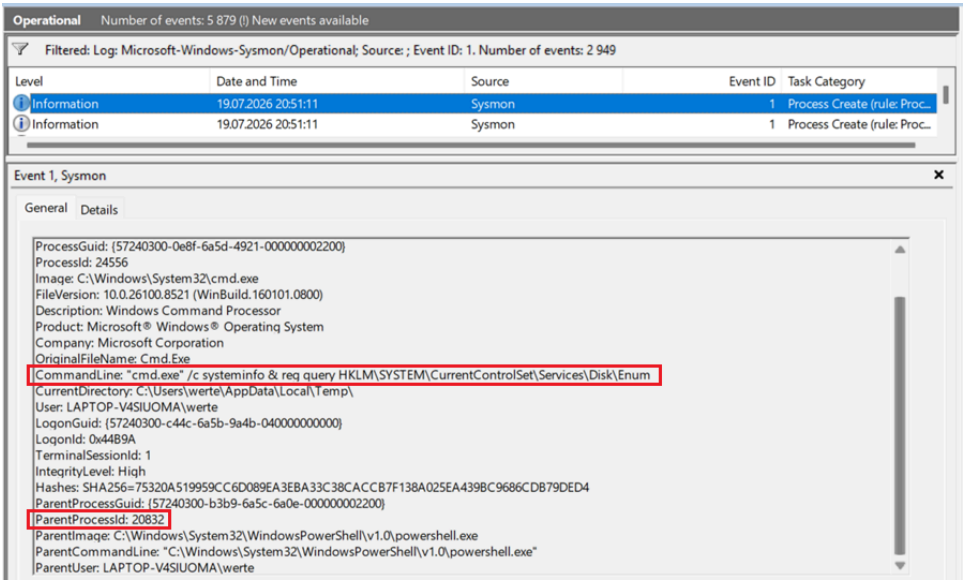


Рис. 7.11. Тест T1082:

cmd.exe з командою `systeminfo & reg query HKLM\SYSTEM\...\Disk\Enum`

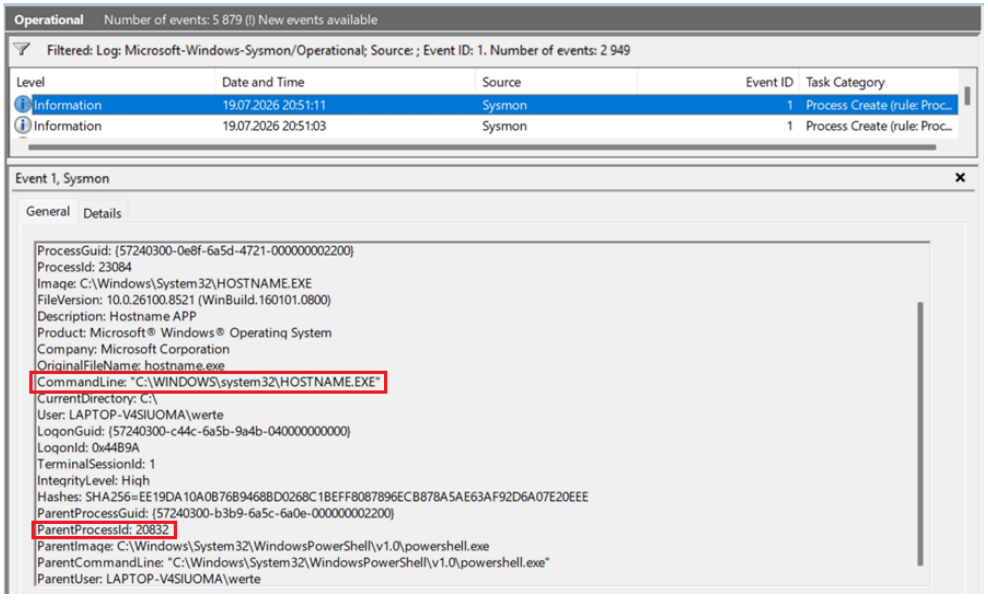


Рис. 7.12. Тест T1082: окремий виклик hostname.exe

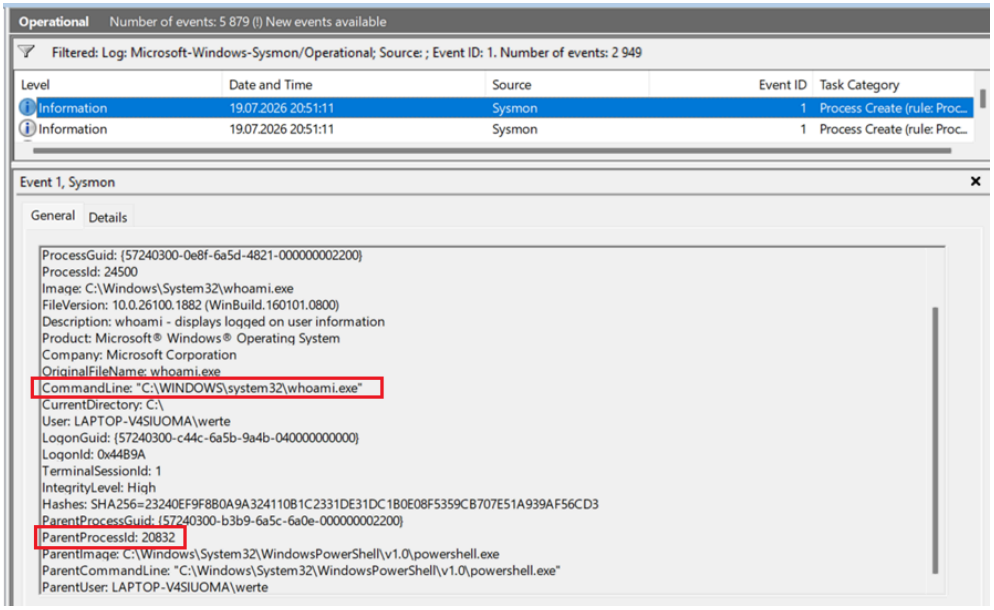


Рис. 7.13. Тест T1082: окремий виклик whoami.exe

Практичний висновок полягає у тому, що назва «атомарний тест» у Atomic Red Team описує одиницю бібліотеки тестів, а не гарантію «один тест = один системний виклик». Для аналітика це означає, що збір доказів для одного номера T-коду Atomic Red Team може вимагати пошуку кількох дочірніх подій одного батьківського процесу, тоді як для Caldera, як правило, достатньо знайти один відповідний запис. Це варто явно відобразити у звіті – порівняльною таблицею «кількість системних викликів на одну техніку» та відповідними скріншотами Sysmon.

Індивідуальні завдання

Кожен здобувач отримує індивідуальний варіант – набір із трьох технік MITRE ATT&CK у межах однієї тактики-фокуса (табл. 7.4).

Завдання виконується лише в Caldera за тим самим принципом, що й демонстраційний набір: 1) знайти в Stockpile здатності для кожної з трьох технік, перевіривши відсутність залежності від Facts / Payload; 2) скласти власний Adversary Profile та виконати операцію; 3) зафіксувати докази в Sysmon для кожної техніки; 4) оформити звіт зі скріншотами.

Таблиця 7.4. Індивідуальні варіанти завдань для здобувачів

№	Тактика-фокус	Набір технік MITRE ATT&CK для складання власного Adversary Profile
1	Discovery	T1082 — System Information Discovery T1057 — Process Discovery T1018 — Remote System Discovery
2	Persistence	T1053.005 — Scheduled Task T1547.001 — Registry Run Keys / Startup Folder T1136.001 — Create Account: Local Account
3	Defense Evasion	T1070.004 — File Deletion (Indicator Removal) T1112 — Modify Registry T1027 — Obfuscated Files or Information
4	Collection	T1005 — Data from Local System T1560.001 — Archive Collected Data T1074.001 — Local Data Staging
5	Credential Access	T1552.001 — Credentials In Files T1087.001 — Account Discovery: Local Account T1016 — System Network Configuration Discovery
6	Discovery / Lateral Movement	T1046 — Network Service Discovery T1135 — Network Share Discovery T1021.001 — Remote Desktop Protocol (*лише виявлення налаштувань, без реального руху*)

Додаткове завдання для всіх варіантів: знайти в Stockpile здатність «File and Directory Discovery» (T1083) з командою `Get-ChildItem -Path #{host.system.path}`, додати в профіль і спробувати виконати. Здатність зависне зі статусом «collect» через незадоволений Requirement (факт `host.system.path`). Завдання: діагностувати причину (вкладка Requirements редактора здатності), спробувати створити факт вручну через розділ Sources і пояснити в звіті результат спроби.

Можливі складнощі та їх вирішення

Наведений перелік складено за результатами реального розгортання стенду на Windows-ноутбуці під час підготовки цієї роботи. Це не вичерпний перелік усіх можливих несправностей, а практичний довідник найімовірніших перешкод для обраної архітектури (Docker Desktop + PowerShell на Windows).

1. «`wsl` не розпізнається як команда»

   Причина: Застаріла збірка Windows або компоненти WSL не увімкнені

   Рішення: Перевірити білд (winver ≥ 19041) і виконати вручну: `dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`

та `VirtualMachinePlatform`, перезавантажити ПК. Docker Desktop часто вмикає ці компоненти сам.

2. «`docker build` падає на кроці плагіна `emu`: «`syntax error: unexpected end of file`» / «`exit code: 2`»»

   Причина: Git for Windows перетворює LF на CRLF у bash-скриптах (core.autocrlf=true) – Linux-контейнер не може виконати такий скрипт

   Рішення: Клонувати репозиторій з явною забороною підміни кінців рядків: `git clone -c core.autocrlf=false https://github.com/mitre/caldera.git --recursive --branch <версія>`

3. «Той самий крок падає на `grep`: «`No such file or directory`»»

   Причина: Файл conf/local.yml відсутній у свіжому клоні (створюється сервером лише при першому запуску, а Docker-збірка читає його вже на етапі build)

   Рішення: Перед `docker build` виконати: `copy conf\default.yml conf\local.yml`

4. «`docker build` обривається: «`failed to receive status: rpc error ... EOF`» після тривалого «unpacking»»

   Причина: Скінчилось вільне місце на диску C: (образ важить ~10+ ГБ; збірка тимчасово займає до 20+ ГБ)

   Рішення: Звільнити місце, прибрати зайвий WSL-дистрибутив (`wsl --unregister Ubuntu`), `docker system prune -a --volumes -f`, за потреби стиснути ext4.vhdx через `diskpart` після `wsl --shutdown`

5. «localhost:8888 не відкривається, хоча контейнер «Up»»

   Причина: Існує кілька контейнерів з того самого образу; активний – той, що запущений без прив'язки портів (`-p`)

   Рішення: `docker ps -a` – знайти контейнер із портами 0.0.0.0:8888 → 8888/tcp, зупинити зайві, `docker start <ім'я>`. Завжди задавати `--name` наперед.

6. «Команда розгортання агента блокується: «This script contains malicious content and has been blocked by your antivirus software»»

   Причина: Windows Defender (AMSI) аналізує текст скрипта до виконання і розпізнає патерн red-team-агента

   Рішення: `Set-MpPreference -DisableRealtimeMonitoring $true` в тому самому вікні перед командою розгортання. Якщо не допомагає (`Get-MpComputerStatus` все ще показує `True`) – вимкнути Tamper Protection вручну: Windows Security → Virus & Threat protection → Manage settings.

7. «Агент у списку Agents має статус «dead»»

   Причина: Sandcat запущений як звичайний процес, а не служба – не переживає перезавантаження хоста чи закриття сесії

   Рішення: Повторно виконати команду розгортання (з'явиться новий paw) – тримати команду напохваті окремим текстовим файлом.

8. «Здатність виконується зі статусом «success», але «No output»»

   Причина: Команда пише результат у файл (>>) замість stdout, або довантажує Sysinternals-утиліту, що чекає підтвердження EULA у неінтерактивному запуску

Рішення: Обирати здатності, де вивід явно повертається в консоль (напр. cat/Get-Content після запису у файл), уникати здатностей, що довантажують сторонні бінарники без /accepteula.

9. «Здатність «зависає» зі статусом «collect», pid: N/A»

   Причина: Здатність має Requirement (наприклад, paw_provenance) на факт, якого немає в операції (позначка замка у стовпці Requires)

   Рішення: Створити факт вручну через Sources, або обрати альтернативну здатність тієї самої техніки без параметризації.

10. «`Install-AtomicRedTeam` падає на «running scripts is disabled on this system»»

    Причина: Execution Policy за замовчуванням (Restricted) забороняє виконання .ps1/.psm1 модулів-залежностей

    Рішення: `Set-ExecutionPolicy Bypass -Scope Process -Force` перед встановленням (діє лише в межах поточного вікна).

Контрольні питання

1. У чому полягає принципова відмінність емуляції супротивника (*adversary emulation*) від традиційного тестування на проникнення (*penetration testing*)?
2. Що таке матриця MITRE ATT&CK і чим тактика (*tactic*) відрізняється від техніки (*technique*)?
3. З яких сутностей складається операція в Caldera (агент, ability, adversary profile, operation) і як вони пов’язані між собою?
4. Чому Atomic Red Team не потребує окремого C2-сервера чи агента для виконання тесту?
5. Що таке Fact у термінології Caldera і чому здатність зі статусом «collect» може ніколи не виконатися?
6. Чому статус «success» не завжди означає, що техніка досягла заявленої мети (наведіть приклад із власного досвіду)?
7. Що таке Purple Team і чому зіставлення дій супротивника з журналом моніторингу Sysmon є основою цієї практики?
8. На прикладі техніки T1082 поясніть, чому один «атомарний» тест Atomic Red Team може розпадатися на кілька окремих подій Process Create, тоді як одна здатність Caldera – зазвичай на одну.

9. У яких практичних ситуаціях доцільніше застосувати Caldera, а в яких – Atomic Red Team? Обґрунтуйте на основі порівняльної таблиці.
10. Чому агент Sandcat не переживає перезавантаження хоста, і які практичні наслідки це має для багатоденної роботи над лабораторною?
11. Чому лабораторний стенд для емуляції супротивника має бути ізольований, і чому в межах цієї роботи (один хост) ця вимога виконана лише частково?

ЛІТЕРАТУРА

1. Mohanty S.N. et al. Protecting and Mitigating Against Cyber Threats: Deploying Artificial Intelligence and Machine Learning. Wiley-Scrivener, 2025. 560 p.
2. Steffens T. Attribution of Advanced Persistent Threats. Springer Vieweg, 2020. 219 p.
3. Patel J. Zero Day Resilience Building Robust Defenses in the Cyber Age. Self Publisher, 2023. 142 p.
4. Crossley C. Software Supply Chain Security. O'Reilly Media, 2024. 242 p.
5. Riggs C. Network Perimeter Security. Auerbach Publications, 2019. 420 p.
6. Mukherjee A. The Complete Guide to Defense in Depth. Packt Publishing, 2024. 298 p.
7. NIST SP 800-207. Zero Trust Architecture. National Institute of Standards and Technology, 2020. 59 p.
8. Moubayed A., Refaey A., Shami A. Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks. *IEEE Network*. 2019. Vol. 33, No. 5. P. 226–233.
9. Relington J.Network Segmentation and Microsegmentation. Independently published, 2025. 209 p.
10. Cybellium. Mastering Microsegmentation: A Comprehensive Guide To Learn Microsegmentation. Independently published, 2023. 124 p.
11. Klein D. Micro-segmentation: securing complex cloud environments. *Network Security*. 2019. Vol. 2019, No. 3. DOI: 10.1016/S1353-4858(19)30034-0
12. Malik P. Machine Learning for Cyber Security. De Gruyter, 2022. 158 p.
13. Chen X. Cyber Security Meets Machine Learning. Springer Verlag, 2021. 163 p.
14. Maurice C., Thompson J., Copeland W. The Foundations of Threat Hunting. Packt Publishing, 2022. 246 p.
15. Costa-Gazcón V. Practical Threat Intelligence and Data-Driven Threat Hunting. Packt Publishing, 2021. 398 p.

16. Benzekki K., El Fergougui A. Elbelrhiti Elalaoui A. Software-defined networking (SDN): a survey. *Security and Communication Networks*. 2016. Vol. 9, No. 18. P. 5803–5833.
17. McKeown N. et al. OpenFlow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*. 2008. Vol. 38, No. 2. P. 69–74.
18. Gray K., Nadeau T.D. Network Function Virtualization. Morgan Kaufmann, 2016. 270 p.
19. Doherty J. SDN and NFV Simplified. Addison-Wesley, 2016. 320 p.
20. Zhang Z., Meddahi A. Security in Network Functions Virtualization. ISTE Press - Elsevier, 2017. 272 p.
21. Firoozjaei M.D. et al. Security challenges with network functions virtualization. Future Generation Computer Systems. 2027. Vol. 67. P. 315–324.
22. Xu K. Network Behavior Analysis: Measurement, Models, and Applications. Springer Nature, 2021. 163 p.
23. Blokdyk G. User and Entity Behavior Analytics (2nd Ed.). 5STARCooks, 2022. 309 p.
24. Spitzner L. Honeypots: Tracking Hackers. Addison Wesley, 2002. 480 p.
25. The Honeynet Project. Know Your Enemy: Learning about Security Threats. Addison Wesley, 2004. 800 p.
26. Understanding DoT and DoH (DNS over TLS vs. DNS over HTTPS). URL: https://www.cloudns.net/blog/understanding-dot-and-doh-dns-over-tls-vs-dns-over-https/ (дата зверненняЖ 20.07.2025).
27. Simpson A. et al. Quick UDP Internet Connections and Transmission Control Protocol in unsafe networks: A comparative analysis. *IET Smart Cities*. 2024. Vol. 6, No. 4. P. 351–360.
28. Cremers C. et al. A Comprehensive Symbolic Analysis of TLS 1.3. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). New York, USA, pp. 1773–1788.
29. Dowling B., Paterson K.G. A Cryptographic Analysis of the WireGuard Protocol. Proceedings of the 16th International Conference of Applied

Cryptography and Network Security (ACNS 2018). Leuven, Belgium, July 2-4, 2018. Springer-Verlag, Berlin, Heidelberg, pp. 3–21.
30. RFC 9114. Internet Engineering Task Force (IETF). Internet Engineering Task Force, 2022.
31. Manson R. Getting Started with WebRTC. Packt Publishing, 2013. 114 p.
32. Indrasiri K., Kuruppu D. gRPC: Up and Running. O'Reilly Media, 2020. 202 p.
33. Madden N. API Security in Action. Manning, 2020. 576 p.
34. Gartner, The Future of Network Security Is in the Cloud, Neil MacDonald, Lawrence Orans, Joe Skorupa, 30 August 2019.
35. Blokdyk G. Cloud Access Security Broker. 5STARCooks, 2021. 297 p.
36. Dotson C. Practical Cloud Security. O'Reilly Media, 2019. 193 p.
37. Climent E.F. Cloud Security Posture Management (CSPM) Demystified. Independently published, 2024. 494 p.
38. Kovacevic B., DiCola N. Security Orchestration, Automation, and Response for Security Analysts. Packt Publishing, 2013. 338 p.
39. Understanding Major Playbooks for Your Favourite SOAR. URL: https://hub.metronlabs.com/understanding-major-playbooks-for-your-favourite-soar/ (дата звернення Ж 20.07.2025).
40. Allsopp W. Advanced Penetration Testing. Wiley, 2017. 288 p.
41. Vogt H. Adversary Simulation: A Guide to Red Team Hacking. Independently published, 2023. 181 p.
42. Blokdyk G. Breach and Attack Simulation. 5STARCooks, 2022. 311 p.
43. Zhukabayeva T. et al. Cybersecurity Solutions for Industrial Internet of Things–Edge Computing Integration: Challenges, Threats, and Future Directions. *Sensors*. 2025. Vol. 25, No. 1. Article No. 213.
44. Xiao Y. et al. Edge Computing Security: State of the Art and Challenges. *Proceedings of the IEEE*. 2019. Vol. 107, No. 8. P. 1608–1631.
45. Multi-Cloud Security: Overview, Benefits, and Examples. URL: https://online.utulsa.edu/blog/multi-cloud-security/ (дата звернення: 15.11.2024).

46. Siva Kumar R.S. et al. Adversarial Machine Learning-Industry Perspectives. Proceedings of the 2020 IEEE Security and Privacy Workshops (SPW), San Francisco, USA, 2020, pp. 69–75.
47. Martínez J., Durán J.M. Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study. *International Journal of Safety and Security Engineering*. 2021. Vol. 11, No. 5. P. 537–545.
48. Singh N., Tripathy S. Unveiling the veiled: An early stage detection of fileless malware. *Computers & Security*. 2025. Vol. 150. Article No. 104231.
49. Adeyeri A., Abroshan H. Geopolitical Ramifications of Cybersecurity Threats: State Responses and International Cooperations in the Digital Warfare Era. *Information*. 2024. Vol. 15, No. 11. Article No. 682.
50. Adamson K.M., Qureshi A. Zero Trust 2.0: Advances, Challenges, and Future Directions in ZTA. URL: https://doi.org/10.21203/rs.3.rs-6602547/v1 (дата звернення: 20.07.2025).
51. Mattei M. Data-Driven Cybersecurity. Manning, 2025. 352 p.
52. Gong Y. et al. Practical solutions in fully homomorphic encryption: a survey analyzing existing acceleration methods. *Cybersecurity*. 2024. Vol. 7. Article No. 5.
53. Mulligan D.P. et al. Confidential Computing — a brave new world. Proceedings of 2021 International Symposium on Secure and Private Execution Environment Design (SEED), Washington, USA, 2021, pp. 132–138.