Практична робота №3

БЕЗПЕКА ТА ЗАХИСТ ВЕБСЕРВЕРА NGINX (SECURITY HARDENING)

Мета заняття: набути практичних навичок налаштування безпеки вебсервера nginx; ознайомитися з поняттям security hardening та його складовими; навчитися приховувати технічну інформацію про вебсервер, налаштовувати HTTP-заголовки безпеки, обмежувати доступ за методами та IP-адресами; засвоїти налаштування SSL/TLS-шифрування з самопідписними сертифікатами та сертифікатами Let's Encrypt через DNS-01 challenge; отримати навички роботи з Fail2Ban та ModSecurity (WAF); навчитися захищати чутливі шляхи, налаштовувати базову автентифікацію та журналювання підозрілої активності.

Завдання на роботу

Підготовка: налаштування мережі Vagrant. Перед виконанням завдань необхідно розширити мережеву конфігурацію VM у `Vagrantfile`. Стандартне налаштування Vagrant використовує лише NAT-інтерфейс, через який VM виходить в Інтернет, та port forwarding (8080 → 80). Це дозволяє звертатися до nginx лише через `localhost:8080` з хост-машини, але не дає можливості адресувати VM за її власною IP-адресою. Додаткова мережа типу **host-only (private_network)** створює окремий мережевий інтерфейс між хост-машиною та VM: VM отримує фіксовану IP-адресу `192.168.56.10`, а хост-машина — `192.168.56.1` (шлюз підмережі). Це необхідно для завдання 3, де демонструється IP-фільтрація: хост-машина звертається до nginx за доменним іменем безпосередньо через цю підмережу та отримує заблокований доступ відповідно до правил `allow/deny`.

Знайти у `Vagrantfile` рядок з `forwarded_port` та додати одразу після нього рядок `private_network` (рис. 1):

```
Vagrant.configure("2") do |config|

  # ... (інші налаштування box, CPU, RAM) ...

  # Порт-форвардинг: localhost:8080 → VM:80, localhost:8443 → VM:443
  config.vm.network "forwarded_port", guest: 80,  host: 8080
  config.vm.network "forwarded_port", guest: 443, host: 8443

  config.vm.network "private_network", ip: "192.168.56.10"
```

Рис. 1 – Фрагмент Vagrantfile з додаванням host-only мережі

Якщо VM ще не запускалась — запустити командою `vagrant up`. Якщо VM вже запущена — застосувати нові мережеві налаштування без повного перезавантаження: `vagrant reload`. Перевірити застосовану конфігурацію, виконавши команду: `vagrant ssh -c "ip a show | grep '192.168.56'"`. Зробити висновки за отриманими результатами.

1. Створити та розгорнути тестовий сайт кав'ярні.

Етап 1 — розробка на хост-машині. Необхідно створити статичний сайт кав'ярні локально на хост-машині (не на VM), ініціалізувати git-репозиторій та опублікувати код на GitHub. Розроблений сайт буде використано у подальших завданнях для демонстрації різних аспектів захисту вебсервера.

a. Створити директорію проєкту та ініціалізувати git-репозиторій на хост-машині. Назва репозиторію: `security-lab-xxx-yyy-zzz`, де `xxx` — назва групи (наприклад, `kim251`), `yyy` — номер варіанту (наприклад, `005`), `zzz` — ініціали студента (наприклад, `mvv`), `xxx-yyy-zzz` тут і далі мають бути однакові:

```
mkdir security-lab-xxx-yyy-zzz && cd security-lab-xxx-yyy-zzz
git init
git remote add origin https://github.com/<username>/security-lab-xxx-yyy-zzz.git
```

b. Створити структуру директорій сайту, виконавши:

```
mkdir -p admin uploads assets/css assets/images
touch uploads/.gitkeep   # порожній каталог для тестування блокування скриптів
```

c. Створити файл `.gitignore` з наступним вмістом (рис. 1.c):

```
# .env — розкоментуйте у реальних проєктах! Тут закоментовано навмисно
# для навчальних цілей: файл залишено у репозиторії для демонстрації
# небезпеки місконфігурації та важливості захисту на рівні вебсервера.
# .env

# .htpasswd — аналогічно, розкоментуйте у реальних проєктах!
# .htpasswd

# Стандартні виключення
node_modules/
*.log
*.tmp
.DS_Store
Thumbs.db
```

Рис. 1.c – Вміст .gitignore

ВАЖЛИВА ПРИМІТКА: у реальних проєктах файли `.env`, `.htpasswd` та директорія `.git/` ОБОВ'ЯЗКОВО додаються до `.gitignore` і НІКОЛИ не потрапляють до репозиторію, оскільки містять чутливу інформацію (паролі, ключі, токени). У цій роботі ці файли свідомо залишені у репозиторії як приклад типової місконфігурації (misconfiguration). У завданні 8 на практиці буде продемонстровано, що без захисту на рівні nginx ці файли доступні будь-якому клієнту — та усунено цей недолік за допомогою відповідних location-блоків.

d. Створити демонстраційний файл `.env` (рис. 1.d):

```
# Демонстраційний файл змінних середовища
# УВАГА: у реальних проєктах — НІКОЛИ не комітити до git-репозиторію!

DB_PASSWORD=demo123
DB_HOST=localhost
DB_PORT=5432
DB_NAME=coffee_shop_db
SECRET_KEY=mysecretkey_demo_only_not_for_production
API_KEY=demo_api_key_12345
STRIPE_SECRET=sk_test_demo_key_here
DEBUG=true
APP_ENV=development
```

Рис. 1.d – Вміст .env

e. Створити демонстраційний файл `.htpasswd` (рис. 1.e):

```
# Демонстраційний файл .htpasswd (не реальний хеш паролю)
# У реальних проєктах генерується командою: htpasswd -c .htpasswd admin
admin:$apr1$rBZT.Hn4$DemoHashForEducationalPurposesOnly
manager:$apr1$xYz1.Ab2$AnotherDemoHashForEducation
```

Рис. 1.e – Вміст .htpasswd

f. Створити головну сторінку `index.html` (рис. 1.f):

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Кав'ярня "Арабіка"</title>
    <link rel="stylesheet" href="/assets/css/style.css">
</head>
<body>
    <header>
        <h1>&#9749; Кав'ярня "Арабіка"</h1>
        <nav>
            <a href="/">Головна</a>
            <a href="/menu.html">Меню</a>
            <a href="/about.html">Про нас</a>
            <a href="/contact.html">Контакти</a>
        </nav>
    </header>
    <main>
        <section class="hero">
            <h2>Ласкаво просимо!</h2>
            <p>Найкраща кава у вашому місті з 2010 року.</p>
            <img src="/assets/images/coffee.jpg" alt="Чашка кави">
        </section>
        <section class="features">
            <div class="feature"><h3>Свіжа кава</h3><p>Зернова
арабіка</p></div>
            <div class="feature"><h3>Затишна атмосфера</h3><p>Wi-Fi та
тиша</p></div>
        </section>
    </main>
    <footer><p>&copy; 2025 Кав'ярня "Арабіка"</p></footer>
</body>
</html>
```

Рис. 1.f – Вміст index.html

g. Створити сторінку меню `menu.html` (рис. 1.g):

```
<!DOCTYPE html><html lang="uk">
<head><meta charset="UTF-8"><title>Меню — Арабіка</title>
<link rel="stylesheet" href="/assets/css/style.css"></head>
<body>
    <header><h1>&#9749; Кав'ярня "Арабіка"</h1>
    <nav><a href="/">Головна</a><a href="/menu.html">Меню</a>
    <a href="/about.html">Про нас</a><a
href="/contact.html">Контакти</a></nav></header>
    <main>
        <h2>Наше меню</h2>
        <section class="menu-section">
            <h3>Кава</h3>
            <ul>
                <li>Еспресо — 45 грн</li>
                <li>Американо — 55 грн</li>
                <li>Капучино — 70 грн</li>
                <li>Латте — 75 грн</li>
                <li>Флет Вайт — 80 грн</li>
            </ul>
        </section>
        <section class="menu-section">
            <h3>Їжа</h3>
            <ul>
                <li>Круасан — 55 грн</li>
                <li>Тост — 65 грн</li>
            </ul>
        </section>
    </main>
    <footer><p>&copy; 2025 Кав'ярня "Арабіка"</p></footer>
</body></html>
```

Рис. 1.g – Вміст menu.html

h. Створити сторінку контактів `contact.html` з HTML-формою (рис. 1.h). Форма використовуватиметься у завданні 7 для тестування XSS-блокування через ModSecurity:

```
<!DOCTYPE html><html lang="uk">
<head><meta charset="UTF-8"><title>Контакти — Арабіка</title>
<link rel="stylesheet" href="/assets/css/style.css"></head>
<body>
    <header><h1>&#9749; Кав'ярня "Арабіка"</h1>
    <nav><a href="/">Головна</a><a href="/menu.html">Меню</a>
    <a href="/about.html">Про нас</a><a
href="/contact.html">Контакти</a></nav></header>
    <main>
        <h2>Зв'яжіться з нами</h2>
        <p>вул. Кавова, 1, Київ | +38 044 000 00 00</p>
        <section class="contact-form">
            <h3>Напишіть нам</h3>
            <!-- Форма без бекенду — для тестування POST/XSS через
ModSecurity -->
            <form method="POST" action="/contact.html">
                <label>Ваше ім'я:
                    <input type="text" name="name" required></label>
                <label>Email:
                    <input type="email" name="email" required></label>
                <label>Повідомлення:
                    <textarea name="message" rows="5"></textarea></label>
                <button type="submit">Надіслати</button>
            </form>
        </section>
    </main>
    <footer><p>&copy; 2025 Кав'ярня "Арабіка"</p></footer>
</body></html>
```

Рис. 1.h – Вміст contact.html з HTML-формою

i. Створити сторінку `admin/index.html` (рис. 1.i). Використовуватиметься у завданні 8 для тестування auth_basic:

```
<!DOCTYPE html><html lang="uk">
<head><meta charset="UTF-8"><title>Адмін — Арабіка</title>
<link rel="stylesheet" href="/assets/css/style.css"></head>
<body>
    <main>
        <h1>Панель управління</h1>
        <p>Доступ лише для адміністратора.</p>
        <ul>
            <li><a href="#">Керування меню</a></li>
            <li><a href="#">Перегляд замовлень</a></li>
            <li><a href="#">Налаштування</a></li>
        </ul>
    </main>
</body></html>
```

Рис. 1.i – Вміст admin/index.html

j. Створити файл стилів `assets/css/style.css` (рис. 1.j):

```
/* assets/css/style.css — стилі кав'ярні "Арабіка" */
* { box-sizing: border-box; margin: 0; padding: 0; }
body { font-family: Arial, sans-serif; background: #f9f5f0; color: #333;
}
header { background: #3e2000; color: #fff; padding: 1rem 2rem;
         display: flex; align-items: center; justify-content: space-
between; }
header h1 { font-size: 1.5rem; }
nav a { color: #ffcf77; text-decoration: none; margin-left: 1.5rem; }
nav a:hover { text-decoration: underline; }
main { max-width: 900px; margin: 2rem auto; padding: 0 1rem; }
.hero { text-align: center; padding: 3rem 1rem; }
.hero h2 { font-size: 2rem; color: #3e2000; margin-bottom: 1rem; }
.hero img { max-width: 300px; border-radius: 8px; margin-top: 1rem; }
.features { display: flex; gap: 1.5rem; margin-top: 2rem; }
.feature { flex: 1; background: #fff; padding: 1.5rem;
           border-radius: 8px; box-shadow: 0 2px 8px rgba(0,0,0,.08); }
form label { display: block; margin-bottom: 1rem; font-weight: bold; }
form input, form textarea { width: 100%; padding: .5rem; margin-top:
.3rem;
    border: 1px solid #ccc; border-radius: 4px; }
button { background: #3e2000; color: #fff; border: none;
         padding: .7rem 2rem; border-radius: 4px; cursor: pointer; }
footer { text-align: center; padding: 1rem; color: #888; font-size:
.9rem; }
```

Рис. 1.j – Вміст assets/css/style.css

k. Додати заповнювач зображення. Скопіювати будь-яке зображення (PNG або JPG) до директорії `assets/images/` та назвати його `coffee.jpg`. Зображення може бути завантажене з будь-якого вільно доступного ресурсу. Закомітити всі файли та опублікувати на GitHub, виконавши:

```
git add .
git status   # переглянути список файлів, що будуть
закомічені
git commit -m "initial commit: coffee shop website
for security hardening lab"
git push -u origin main
```

Етап 2 — розгортання на VM. Необхідно клонувати репозиторій на віртуальну машину Ubuntu та налаштувати базовий nginx server block для обслуговування сайту.

m. Запустити VM та підключитися через SSH. Якщо `git` не встановлено — встановити:

```
vagrant up && vagrant ssh
sudo apt update && sudo apt install git -y
```

n. Клонувати репозиторій на VM та встановити права, виконавши:

```
sudo git clone
https://github.com/<username>/security-lab-xxx-yyy-
zzz.git \
    /var/www/security-lab-xxx-yyy-zzz
sudo chown -R www-data:www-data /var/www/security-
lab-xxx-yyy-zzz
ls -la /var/www/security-lab-xxx-yyy-zzz
```

o. Створити конфігурацію сайту /etc/nginx/sites-available/security-lab-xxx-yyy-zzz (рис. 1.o):

```
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local;

    root /var/www/security-lab-xxx-yyy-zzz;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Рис. 1.o – Базовий nginx server block для сайту кав'ярні

p. Активувати сайт, перевірити конфігурацію та перезавантажити nginx, виконавши:

```
sudo ln -s /etc/nginx/sites-available/security-lab-
xxx-yyy-zzz \
    /etc/nginx/sites-enabled/
sudo nginx -t
sudo nginx -s reload
```

q. Додати запис у файл `/etc/hosts` на VM, щоб дозволити звертатися до сайту за доменним іменем без DNS. Ця команда виконується один раз і діє для всіх наступних завдань. Перевірити, що запис відсутній перед додаванням:

```
grep "security-lab-xxx-yyy-zzz.local" /etc/hosts ||
sudo sh -c 'echo "127.0.0.1  security-lab-xxx-yyy-
zzz.local" >> /etc/hosts'
cat /etc/hosts | tail -5   # переконатись що запис
додано
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}"
http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

2. Виконати базовий security hardening вебсервера nginx.

Security hardening — процес зміцнення налаштувань системи з метою зменшення площі атаки (attack surface). На рівні nginx це включає приховування технічних відомостей про сервер, встановлення заголовків безпеки, обмеження дозволених методів та вимкнення непотрібних функцій.

a. Приховати версію nginx. Знайти блок `http` у файлі `/etc/nginx/nginx.conf` та додати директиву `server_tokens off` (рис. 2.a). Без цього налаштування nginx додає версію до заголовка `Server:` та до HTML-сторінок помилок, що полегшує пошук відомих вразливостей:

```
http {
    # Приховати версію nginx у заголовку Server та сторінках помилок
    server_tokens off;

    # ... решта налаштувань без змін ...
}
```

Рис. 2.a – Директива server_tokens off у блоці http

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -I http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

b. Створити сніпет `/etc/nginx/snippets/security-headers.conf` з заголовками безпеки (рис. 2.b). Заголовки безпеки — це HTTP-заголовки відповіді, що інструктують браузер клієнта щодо поведінки при відображенні ресурсів:

```
sudo vi /etc/nginx/snippets/security-headers.conf
```

```
# /etc/nginx/snippets/security-headers.conf

# X-Content-Type-Options: забороняє браузеру "вгадувати" MIME-тип
контенту.
# Захищає від MIME-type confusion атак.
add_header X-Content-Type-Options "nosniff" always;

# X-Frame-Options: забороняє вбудовування сторінки в <iframe>.
# Захищає від clickjacking — прихованого клацання підкладеним фреймом.
add_header X-Frame-Options "SAMEORIGIN" always;

# X-XSS-Protection: вмикає XSS-фільтр у старих браузерах (IE, Safari).
# Сучасні браузери використовують CSP замість цього заголовка.
add_header X-XSS-Protection "1; mode=block" always;

# Content-Security-Policy: визначає, звідки браузер може завантажувати
# скрипти, стилі, зображення та інші ресурси. Найпотужніший захист від
XSS.
add_header Content-Security-Policy
    "default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-
inline';
     img-src 'self' data:; font-src 'self';" always;

# Referrer-Policy: контролює, яка інформація надсилається у заголовку
Referer.
# strict-origin-when-cross-origin: надсилати лише origin при крос-
доменних запитах.
add_header Referrer-Policy "strict-origin-when-cross-origin" always;

# Permissions-Policy (раніше Feature-Policy): обмежує доступ до
браузерних API.
add_header Permissions-Policy "camera=(), microphone=(), geolocation=()"
always;
```

Рис. 2.b – Вміст /etc/nginx/snippets/security-headers.conf

c. Оновити server block сайту, підключивши сніпет заголовків безпеки, вимкнувши `autoindex` та обмеживши дозволені HTTP-методи (рис. 2.c):

```
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local;

    root /var/www/security-lab-xxx-yyy-zzz;
    index index.html;

    # Вимкнути відображення вмісту директорій.
    # Без цього nginx може показати список файлів при відсутності
index.html.
    autoindex off;

    # Підключити заголовки безпеки з окремого сніпету
    include /etc/nginx/snippets/security-headers.conf;

    # Дозволити лише безпечні HTTP-методи (GET, HEAD, POST).
    # Методи PUT, DELETE, TRACE, OPTIONS, CONNECT — заблокувати.
    # TRACE може використовуватись для крадіжки cookie (XST-атака).
    if ($request_method !~ ^(GET|HEAD|POST)$) {
        return 405;
    }

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Рис. 2.c – Оновлений server block з security headers та обмеженням HTTP-методів

d. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

e. Перевірити застосування заголовків безпеки, виконавши:

```
curl -I http://security-lab-xxx-yyy-zzz.local
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -I http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

f. Перевірити блокування небезпечних HTTP-методів, виконавши:

```
curl -X DELETE http://security-lab-xxx-yyy-zzz.local
curl -X TRACE http://security-lab-xxx-yyy-zzz.local
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}" -X DELETE \
http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

g. Перевірити, що `autoindex` вимкнено. Зробити тимчасову директорію та спробувати отримати її список:

```
sudo mkdir /var/www/security-lab-xxx-yyy-zzz/testdir
curl http://security-lab-xxx-yyy-zzz.local/testdir/
sudo rmdir /var/www/security-lab-xxx-yyy-zzz/testdir
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}"
http://security-lab-xxx-yyy-zzz.local/testdir/
```

Зробити висновки за отриманих результатів.

3. Налаштувати обмеження доступу та захист від мережевих атак.

Основні типи мережевих атак на рівні вебсервера: DDoS (Distributed Denial of Service) — флуд запитів з метою перевантаження сервера; brute force — перебір паролів або токенів; Slowloris та slow HTTP — надсилання запитів дуже повільно для утримання з'єднань. Nginx надає вбудовані механізми захисту від цих атак.

a. Оголосити зони для rate limiting та connection limiting у блоці `http` файлу `/etc/nginx/nginx.conf` (рис. 3.a). Зони зберігають стан у shared memory — доступний усім worker-процесам nginx:

```
http {
    # ── Rate Limiting (обмеження частоти запитів) ──────────────────
    #
    # $binary_remote_addr — ключ: IP-адреса клієнта (у бінарному форматі,
    # займає 4 байти для IPv4 замість до 15 байт для рядка)
    # zone=general:10m — назва зони та обсяг пам'яті (10 МБ ≈ 160 000 IP)
    # rate=10r/s — максимальна частота: 10 запитів/секунду з однієї IP
    limit_req_zone $binary_remote_addr zone=general:10m rate=10r/s;

    # Суворіша зона для захисту від brute force: 1 запит/секунду
    limit_req_zone $binary_remote_addr zone=auth:10m    rate=1r/s;

    # ── Connection Limiting (обмеження кількості з'єднань) ─────────
    # Захищає від атак з великою кількістю одночасних з'єднань
    limit_conn_zone $binary_remote_addr zone=conn_limit:10m;

    # ... решта налаштувань без змін ...
}
```

Рис. 3.a – Зони rate limiting та connection limiting у блоці http

b. Оновити server block: застосувати rate limiting, connection limiting, обмеження розміру запиту та таймаути захисту від slow HTTP-атак (рис. 3.b):

```
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local;

    # ── Обмеження розміру тіла запиту ──────────────────────────
    # Захищає від завантаження надмірно великих файлів (DoS через диск)
    client_max_body_size 1m;

    # ── Таймаути (захист від Slowloris та slow HTTP-атак) ──────
    # client_body_timeout: час очікування між пакетами тіла запиту
    client_body_timeout   10s;
    # client_header_timeout: час очікування HTTP-заголовків клієнта
    client_header_timeout 10s;
    # send_timeout: час між двома записами у відповідь клієнту
    send_timeout          10s;
    # keepalive_timeout: тривалість keep-alive з'єднання
    keepalive_timeout     30s;

    # ── Rate Limiting ──────────────────────────────────────────
    # burst=20: дозволяє черга до 20 запитів; nodelay: не затримувати
    limit_req zone=general burst=20 nodelay;

    # ── Connection Limiting ────────────────────────────────────
    limit_conn conn_limit 20;

    root  /var/www/security-lab-xxx-yyy-zzz;
    index index.html;
    autoindex off;
    include /etc/nginx/snippets/security-headers.conf;
    if ($request_method !~ ^(GET|HEAD|POST)$) { return 405; }

    location / { try_files $uri $uri/ =404; }

    location /admin/ {
        # Суворіший rate limit для адмін-панелі: 1 запит/с, burst=3
        limit_req zone=auth burst=3 nodelay;
        try_files $uri $uri/ =404;
    }
}
```

Рис. 3.b – Server block з rate limiting, connection limiting та таймаутами

c. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

d. Перевірити rate limiting. Встановити `apache2-utils` (якщо не встановлено), та виконати навантажувальний тест:

```
sudo apt install apache2-utils -y
# Надіслати 50 запитів (10 паралельних) та
спостерігати за кодами відповідей
```

```
ab -n 50 -c 10 \
    http://security-lab-xxx-yyy-zzz.local/
# Переглянути журнал помилок nginx
sudo tail -20 /var/log/nginx/error.log | grep
"limiting"
```

Перевірити застосовану конфігурацію, виконавши команду:

```
ab -n 50 -c 10 http://security-lab-xxx-yyy-
zzz.local/
```

Зробити висновки за отриманими результатами.

е. Продемонструвати обмеження доступу за IP-адресою з використанням host-only мережі (налаштованої у підготовчому кроці). Тимчасово додати до server block директиви `allow/deny` (рис. 3.е), що дозволяють доступ лише з самої VM (localhost `127.0.0.1`) та блокують усіх інших, включаючи хост-машину (`192.168.56.1`):

```
# Тимчасовий блок IP-фільтрації — додати у server block для демонстрації
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local;

    # Дозволити доступ лише з самої VM (localhost)
    allow 127.0.0.1;

    # Заблокувати всі інші IP — включаючи хост-машину (192.168.56.1)
    deny all;

    root  /var/www/security-lab-xxx-yyy-zzz;
    index index.html;
    # ... (решта директив без змін) ...
}
```

Рис. 3.е – Тимчасова IP-фільтрація — дозволено лише localhost VM

Перевірка з VM (доступ дозволено). Виконати на VM через SSH:

```
curl -s -o /dev/null -w "З VM: %{http_code}\n"
  http://security-lab-xxx-yyy-zzz.local
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}"
  http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

Підготовка хост-машини. На хост-машині (не на VM) додати запис у `/etc/hosts` (Linux/macOS) або `C:\\Windows\\System32\\drivers\\etc\\hosts` (Windows), щоб домен вказував на IP VM у host-only мережі:

```
# На Linux/macOS хост-машині (виконується у звичайному терміналі, не в SSH):
echo "192.168.56.10  security-lab-xxx-yyy-zzz.local" | sudo tee -a /etc/hosts
```

Перевірка з хост-машини (доступ заблоковано). Виконати у звичайному терміналі хост-машини (не через vagrant ssh):

```
# На хост-машині:
curl -s -o /dev/null -w "З хоста: %{http_code}\n" http://security-lab-xxx-yyy-zzz.local
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -s -o /dev/null -w "%{http_code}" http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

Відновлення конфігурації. Видалити директиви `allow/deny` зі server block та перезавантажити nginx. Також видалити тимчасовий запис з `/etc/hosts` хост-машини:

```
# На VM — видалити allow/deny з nginx конфігурації, потім:
sudo nginx -t && sudo nginx -s reload
# На хост-машині — видалити тимчасовий рядок з /etc/hosts:
sudo sed -i '/security-lab-xxx-yyy-zzz.local/d' /etc/hosts
```

4. Налаштувати SSL/TLS з самопідписним сертифікатом.

Самопідписний сертифікат (self-signed certificate) — сертифікат X.509, підписаний власним приватним ключем, а не довіреним центром сертифікації (CA). Він забезпечує шифрування трафіку між клієнтом і сервером, але браузери відображають попередження, оскільки не можуть перевірити автентичність. Підходить для локальних та навчальних середовищ. Параметри SSL/TLS значно впливають на безпеку: застарілі протоколи SSLv3, TLSv1.0, TLSv1.1 та слабкі шифри не слід використовувати.

a. Створити директорію для SSL-матеріалів та згенерувати самопідписний сертифікат, виконавши:

```
sudo mkdir -p /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey
rsa:2048 \
    -keyout /etc/nginx/ssl/security-lab.key \
    -out    /etc/nginx/ssl/security-lab.crt \
    -subj
"/C=UA/ST=Kyiv/L=Kyiv/O=University/OU=IT/CN=security
-lab-xxx-yyy-zzz.local"
# Встановити правильні права: ключ — лише для root,
сертифікат — для читання
sudo chmod 600 /etc/nginx/ssl/security-lab.key
sudo chmod 644 /etc/nginx/ssl/security-lab.crt
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo openssl x509 -in /etc/nginx/ssl/security-
lab.crt -noout -subject -dates
```

Зробити висновки за отриманими результатами.

b. Замінити вміст файлу `/etc/nginx/sites-available/security-lab-xxx-yyy-zzz` конфігурацією HTTPS (рис. 4.b). Параметри TLS налаштовано відповідно до рекомендацій Mozilla Modern:

```
# Блок HTTP → перенаправляє на HTTPS (постійний редирект 301)
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local;
    return 301 https://$host$request_uri;
}

# Блок HTTPS
server {
    listen 443 ssl;
    server_name security-lab-xxx-yyy-zzz.local;

    # ── SSL-матеріали ──────────────────────────────────────────
    ssl_certificate     /etc/nginx/ssl/security-lab.crt;
    ssl_certificate_key /etc/nginx/ssl/security-lab.key;

    # ── Протоколи TLS (лише TLS 1.2 та 1.3 — безпечні) ─────────
    # SSLv3, TLSv1.0, TLSv1.1 — вразливі (POODLE, BEAST, CRIME тощо)
    ssl_protocols TLSv1.2 TLSv1.3;

    # ── Набір шифрів (Mozilla Modern Compatibility) ────────────
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256
                :ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384
                :ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305;

    # Для TLS 1.3 вибір шифру завжди на стороні клієнта; off — правильно
    ssl_prefer_server_ciphers off;

    # ── Кешування сесій (зменшення навантаження TLS handshake) ─
    ssl_session_cache   shared:SSL:10m;  # shared між worker-процесами
    ssl_session_timeout 1d;              # кешувати до 1 доби
    ssl_session_tickets off;             # вимкнути (проблема з PFS)

    # ── HSTS (HTTP Strict Transport Security) ──────────────────
    # Наказує браузеру завжди використовувати HTTPS для цього домену.
    # max-age=63072000 = 2 роки (рекомендований мінімум)
    add_header Strict-Transport-Security
        "max-age=63072000; includeSubDomains" always;

    client_max_body_size 1m;
    client_body_timeout 10s;  client_header_timeout 10s;
    send_timeout 10s;         keepalive_timeout 30s;
    limit_req  zone=general burst=20 nodelay;
    limit_conn conn_limit 20;

    root  /var/www/security-lab-xxx-yyy-zzz;
    index index.html;
    autoindex off;
    include /etc/nginx/snippets/security-headers.conf;
    if ($request_method !~ ^(GET|HEAD|POST)$) { return 405; }

    location / { try_files $uri $uri/ =404; }
}
```

Рис. 4.b – Конфігурація nginx з SSL/TLS та редиректом HTTP→HTTPS

c. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

d. Перевірити редирект HTTP→HTTPS, виконавши:

```
curl -I http://security-lab-xxx-yyy-zzz.local
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -I http://security-lab-xxx-yyy-zzz.local
```

Зробити висновки за отриманими результатами.

e. Перевірити HTTPS та параметри TLS. Прапорець `-k` потрібен для самопідписного сертифікату (обхід перевірки CA):

```
curl -k -I https://security-lab-xxx-yyy-zzz.local

# Перевірити версію TLS та шифр
openssl s_client -connect 127.0.0.1:443 \
    -servername security-lab-xxx-yyy-zzz.local
</dev/null 2>&1 \
    | grep -E "Protocol|Cipher|Verify|subject"
```

Перевірити застосовану конфігурацію, виконавши команду:

```
openssl s_client -connect 127.0.0.1:443 -servername
security-lab-xxx-yyy-zzz.local </dev/null 2>&1 |
grep Protocol
```

Зробити висновки за отриманими результатами.

5. Отримати сертифікат Let's Encrypt через Certbot з DNS-01 validation.

Let's Encrypt — автоматизований безкоштовний CA (Certificate Authority), що видає довірені сертифікати X.509. Браузери довіряють цим сертифікатам без попередження. Сертифікати дійсні 90 днів і поновлюються автоматично через Certbot.

Чому DNS-01, а не HTTP-01? HTTP-01 challenge перевіряє права на домен через HTTP-запит до файлу на сервері (.well-known/acme-challenge/). Це вимагає, щоб сервер був доступний з Інтернету на порту 80. Оскільки сайт розгорнуто на локальній VM у VirtualBox без зовнішнього доступу, HTTP-01 неможливий. DNS-01 challenge перевіряє права на домен через TXT-запис у DNS-зоні — це не потребує доступу до сервера ззовні. Плагін certbot-dns-cloudflare автоматизує додавання/видалення TXT-записів через Cloudflare API.

Підготовчий крок 1 — реєстрація домену на nic.ua. Перейти на сайт `https://nic.ua` та зареєструватись. У розділі реєстрації доменів обрати зону `.pp.ua` та зареєструвати домен виду `security-lab-xxx-yyy-zzz.pp.ua`. Домени у зоні .pp.ua доступні безкоштовно для фізичних осіб на умовах nic.ua. Підтвердити email та зачекати завершення реєстрації (зазвичай кілька хвилин).

Підготовчий крок 2 — підключення до Cloudflare. Зареєструватись на `https://cloudflare.com`. На головній панелі обрати "Add a Site" та ввести щойно зареєстрований домен. Cloudflare просканує існуючі DNS-записи та запропонує план — обрати безкоштовний (Free). Після цього Cloudflare надасть NS-записи виду `emma.ns.cloudflare.com` та `karl.ns.cloudflare.com` (назви можуть відрізнятись). Перейти до панелі nic.ua, знайти свій домен та замінити NS-записи реєстратора на надані Cloudflare. Поширення DNS-змін займає від 5 хвилин до 48 годин (зазвичай до 30 хвилин). Перевірити готовність командою:

```
nslookup -type=NS security-lab-xxx-yyy-zzz.pp.ua 8.8.8.8
```

Підготовчий крок 3 — створення API-токену Cloudflare. У панелі Cloudflare перейти до My Profile → API Tokens → Create Token. Обрати шаблон *"Edit zone DNS"*. У полі "Zone Resources" обрати "Specific zone" → ваш домен. Зберегти токен — він відображається лише один раз. Якщо токен втрачено — необхідно створити новий.

а. Встановити Certbot та плагін для DNS через Cloudflare, виконавши:

```
sudo apt update
sudo apt install certbot python3-certbot-dns-cloudflare
```

Перевірити застосовану конфігурацію, виконавши команду:

```
certbot --version
```

Зробити висновки за отриманими результатами.

b. Створити захищений файл з API-токеном Cloudflare (рис. 5.b):

```
sudo mkdir -p /root/.secrets/certbot
sudo chmod 700 /root/.secrets/certbot
sudo nano /root/.secrets/certbot/cloudflare.ini
```

```
# /root/.secrets/certbot/cloudflare.ini
# Токен API Cloudflare для управління DNS-записами.
# Права токену: Zone:Zone:Read, Zone:DNS:Edit
# Обмежити лише своєю зоною у налаштуваннях Cloudflare!

dns_cloudflare_api_token = ВАШ_CLOUDFLARE_API_TOKEN_ТУТ
```

Рис. 5.b – Вміст /root/.secrets/certbot/cloudflare.ini

```
# Встановити суворі права — файл лише для root
sudo chmod 600 /root/.secrets/certbot/cloudflare.ini
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo ls -la /root/.secrets/certbot/
```

Зробити висновки за отриманими результатами.

c. Отримати SSL/TLS-сертифікат через DNS-01 challenge. Certbot автоматично додасть TXT-запис `_acme-challenge.<domain>` у DNS-зону через Cloudflare API, дочекається підтвердження від Let's Encrypt CA, отримає сертифікат та видалить TXT-запис (рис. 5.c):

```
# Отримання сертифікату через DNS-01 challenge
sudo certbot certonly \
    --dns-cloudflare \
    --dns-cloudflare-credentials /root/.secrets/certbot/cloudflare.ini \
    --dns-cloudflare-propagation-seconds 30 \
    -d security-lab-xxx-yyy-zzz.pp.ua \
    --preferred-challenges dns-01 \
    --non-interactive \
    --agree-tos \
    -m student@example.com

# Параметри:
#   --dns-cloudflare-propagation-seconds 30
#       час очікування після додавання TXT-запису перед перевіркою
#   --non-interactive  — не задавати питань (для автоматизації)
#   --agree-tos        — погодитись з умовами використання
```

Рис. 5.c – Команда отримання сертифікату Let's Encrypt через DNS-01

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo certbot certificates
```

Зробити висновки за отриманими результатами.

d. Оновити конфігурацію nginx для використання сертифікату Let's Encrypt (рис. 5.d):

```
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.pp.ua;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name security-lab-xxx-yyy-zzz.pp.ua;

    # ── Сертифікат Let's Encrypt ──────────────────────────────
    # fullchain.pem = сертифікат сайту + ланцюжок CA-сертифікатів
    ssl_certificate
        /etc/letsencrypt/live/security-lab-xxx-yyy-
zzz.pp.ua/fullchain.pem;
    ssl_certificate_key
        /etc/letsencrypt/live/security-lab-xxx-yyy-zzz.pp.ua/privkey.pem;

    # Рекомендовані параметри від Certbot (TLS протоколи, шифри, DH-
параметри)
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    add_header Strict-Transport-Security
        "max-age=63072000; includeSubDomains" always;

    client_max_body_size 1m;
    client_body_timeout 10s;  client_header_timeout 10s;
    send_timeout 10s;         keepalive_timeout 30s;
    limit_req  zone=general burst=20 nodelay;
    limit_conn conn_limit 20;

    root  /var/www/security-lab-xxx-yyy-zzz;
    index index.html;
    autoindex off;
    include /etc/nginx/snippets/security-headers.conf;
    if ($request_method !~ ^(GET|HEAD|POST)$) { return 405; }

    location / { try_files $uri $uri/ =404; }
}
```

Рис. 5.d – Конфігурація nginx з сертифікатом Let's Encrypt

e. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -I https://security-lab-xxx-yyy-zzz.pp.ua
```

Зробити висновки за отриманими результатами.

f. Перевірити та налаштувати автоматичне оновлення сертифікату, виконавши:

```
# Перевірити статус systemd-таймера Certbot
sudo systemctl status certbot.timer
sudo systemctl list-timers | grep certbot
# Якщо таймер неактивний — активувати
sudo systemctl enable certbot.timer
sudo systemctl start certbot.timer
```

g. Виконати тестове оновлення (dry-run) для перевірки конфігурації, виконавши:

```
sudo certbot renew --dry-run
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo certbot renew --dry-run
```

Зробити висновки за отриманими результатами.

6. Налаштувати захист від брутфорс-атак та сканерів за допомогою Fail2Ban.

Fail2Ban — демон захисту, що аналізує лог-файли у реальному часі та автоматично блокує IP-адреси зловмисників через `iptables/nftables`. Принцип роботи: парсер аналізує рядки журналів за регулярними виразами (фільтрами); якщо кількість збігів для одного IP перевищує `maxretry` за час `findtime`, IP блокується на `bantime` секунд. Fail2Ban використовує концепцію *jail* (в'язниця) — набір параметрів: лог-файл, фільтр, bantime, findtime, maxretry.

a. Встановити Fail2Ban, виконавши:

```
sudo apt update && sudo apt install fail2ban -y
sudo systemctl enable fail2ban && sudo systemctl
start fail2ban
sudo systemctl status fail2ban
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo fail2ban-client ping
```

Зробити висновки за отриманими результатами.

b. Переглянути стандартну конфігурацію та доступні фільтри. Ніколи не редагувати `jail.conf` безпосередньо — він перезаписується при оновленні. Власні налаштування — у файлі `/etc/fail2ban/jail.d/`:

```
# Доступні вбудовані фільтри для nginx
ls /etc/fail2ban/filter.d/ | grep nginx
# Переглянути вбудований фільтр nginx-http-auth
cat /etc/fail2ban/filter.d/nginx-http-auth.conf
```

c. Створити конфігурацію jail для nginx у `/etc/fail2ban/jail.d/nginx-security-lab.conf` (рис. 6.c):

```
sudo nano /etc/fail2ban/jail.d/nginx-security-lab.conf
```

```
# /etc/fail2ban/jail.d/nginx-security-lab.conf

# ── Стандартні nginx jails (вже є вбудовані фільтри) ──────────

[nginx-http-auth]
# Блокує IP при множинних помилках автентифікації (401)
enabled  = true
port     = http,https
logpath  = %(nginx_error_log)s
maxretry = 3
bantime  = 600      # 10 хвилин
findtime = 300      # 5 хвилин

[nginx-botsearch]
# Блокує сканери вразливостей (пошук phpMyAdmin, wp-login тощо)
enabled  = true
port     = http,https
logpath  = %(nginx_access_log)s
maxretry = 5
bantime  = 3600     # 1 година
findtime = 600
filter   = nginx-botsearch

# ── Кастомний jail для нашого сайту ──────────────────────────

[nginx-security-lab]
# Блокує IP, що намагаються отримати доступ до чутливих файлів
enabled  = true
port     = http,https
logpath  = /var/log/nginx/access.log
maxretry = 3
bantime  = 1800     # 30 хвилин
findtime = 300      # 5 хвилин
filter   = nginx-security-lab
```

Рис. 6.c – Вміст /etc/fail2ban/jail.d/nginx-security-lab.conf

d. Створити кастомний фільтр для виявлення підозрілих запитів у `/etc/fail2ban/filter.d/nginx-security-lab.conf` (рис. 6.d): `sudo nano /etc/fail2ban/filter.d/nginx-security-lab.conf`

```
# /etc/fail2ban/filter.d/nginx-security-lab.conf
# Регулярний вираз для виявлення спроб доступу до чутливих шляхів.
# <HOST> — Fail2Ban автоматично замінює на IP-адресу з лог-рядка.

[INCLUDES]
before = common.conf

[Definition]
failregex = ^<HOST> .* "(GET|POST|HEAD)
    \/(\.env|\.git|\.htpasswd|\.ssh|admin\.php
    |wp-admin|wp-login|phpmyadmin|shell\.php|eval\.php
    |config\.php|xmlrpc\.php|\.DS_Store)
    .* HTTP\/.*" (400|403|404) .*$

            ^<HOST> .* "(GET|POST) .* HTTP\/.*" 400 .*$

ignoreregex =
```

Рис. 6.d – Вміст /etc/fail2ban/filter.d/nginx-security-lab.conf

e. Перезавантажити Fail2Ban та перевірити статус, виконавши:

```
sudo systemctl restart fail2ban
sudo fail2ban-client status
sudo fail2ban-client status nginx-security-lab
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo fail2ban-client status nginx-security-lab
```

Зробити висновки за отриманими результатами.

f. Перевірити синтаксис фільтру проти реального лог-файлу, виконавши:

```
sudo fail2ban-regex /var/log/nginx/access.log \
    /etc/fail2ban/filter.d/nginx-security-lab.conf
```

g. Провести тестування: згенерувати підозрілі запити для спрацювання Fail2Ban, виконавши:

```
# Надіслати серію підозрілих запитів (перевищить
maxretry=3)
for i in $(seq 1 10); do
    curl -s http://security-lab-xxx-yyy-
zzz.local/.env > /dev/null
    curl -s http://security-lab-xxx-yyy-
zzz.local/.git/config > /dev/null
    curl -s http://security-lab-xxx-yyy-
zzz.local/wp-admin/ > /dev/null
    curl -s http://security-lab-xxx-yyy-
zzz.local/phpmyadmin/ > /dev/null
    sleep 0.5
done
# Спостерігати за журналом Fail2Ban у реальному часі
(в окремому вікні/вкладці терміналу)
sudo tail -f /var/log/fail2ban.log
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo fail2ban-client status nginx-security-lab
```

Зробити висновки за отриманими результатами.

h. Розблокувати власну IP-адресу після тестування, виконавши:

```
sudo fail2ban-client set nginx-security-lab unbanip
127.0.0.1
sudo fail2ban-client status nginx-security-lab   #
перевірити
```

7. Встановити та налаштувати ModSecurity — Web Application Firewall (WAF).

WAF (Web Application Firewall) аналізує HTTP/HTTPS-трафік та блокує вебатаки на рівні застосунку (OSI Layer 7): SQL-ін'єкції (SQLi), міжсайтовий скриптинг (XSS), Remote File Inclusion (RFI), Path Traversal тощо. Це доповнення до мережевого брандмауера, що працює на рівні мережі (Layer 3-4). ModSecurity — найпоширеніший відкритий WAF для nginx (у вигляді модуля libnginx-mod-http-modsecurity). OWASP Core Rule Set (CRS) —

стандартний набір правил, що підтримується OWASP Foundation та покриває більшість відомих вебатак.

a. Встановити модуль ModSecurity та набір правил OWASP CRS, виконавши:

```
sudo apt update
sudo apt install libnginx-mod-http-modsecurity
modsecurity-crs -y
# Перевірити наявність модуля
ls /usr/lib/nginx/modules/ | grep modsecurity
# Переглянути правила CRS
ls /usr/share/modsecurity-crs/rules/ | head -15
```

Перевірити застосовану конфігурацію, виконавши команду:

```
ls /usr/lib/nginx/modules/ | grep modsecurity
```

Зробити висновки за отриманими результатами.

b. Перевірити, що модуль підключено до nginx (Ubuntu автоматично створює symlink у `/etc/nginx/modules-enabled/`):

```
ls /etc/nginx/modules-enabled/ | grep modsecurity
cat /etc/nginx/modules-enabled/50-mod-http-
modsecurity.conf 2>/dev/null
```

Якщо symlink відсутній — створити вручну:

```
sudo ln -s /etc/nginx/modules-available/50-mod-http-
modsecurity.conf \
    /etc/nginx/modules-enabled/
```

c. Налаштувати базову конфігурацію ModSecurity. Скопіювати та відредагувати шаблон:

```
sudo mkdir -p /etc/nginx/modsecurity

sudo cp /etc/modsecurity/modsecurity.conf-recommended \

  /etc/nginx/modsecurity/modsecurity.conf
```

Змінити режим на `DetectionOnly` (початковий — лише виявлення, без блокування). Переглянути файл та переконатись у ключових налаштуваннях (рис. 7.c):

```
# /etc/nginx/modsecurity/modsecurity.conf (ключові параметри)

# Режим: DetectionOnly — виявлення без блокування (для початку)
# Пізніше замінити на "SecRuleEngine On" для активного захисту
SecRuleEngine DetectionOnly

# Перевіряти тіло POST-запиту (потрібно для виявлення XSS у формах)
SecRequestBodyAccess On
SecRequestBodyLimit 13107200          # 12.5 МБ
SecRequestBodyNoFilesLimit 131072     # 128 КБ (без файлів)

# Журналювання подій безпеки у форматі JSON
SecAuditEngine RelevantOnly
SecAuditLog /var/log/nginx/modsec_audit.log
SecAuditLogFormat JSON

# Дії за замовчуванням (в режимі DetectionOnly — pass, тобто пропустити)
SecDefaultAction "phase:1,log,auditlog,pass"
SecDefaultAction "phase:2,log,auditlog,pass"
```

Рис. 7.c – Ключові параметри /etc/nginx/modsecurity/modsecurity.conf

d. Створити файл підключення ModSecurity + OWASP CRS `/etc/nginx/modsecurity/crs-setup.conf` (рис. 7.d):

```
sudo nano /etc/nginx/modsecurity/crs-setup.conf
```

```
# /etc/nginx/modsecurity/crs-setup.conf
# Головний файл конфігурації — підключає ModSecurity та правила CRS

# 1. Основна конфігурація ModSecurity
Include /etc/nginx/modsecurity/modsecurity.conf

# 2. Налаштування OWASP CRS (рівень паранойї, виключення тощо)
Include /usr/share/modsecurity-crs/crs-setup.conf.example

# 3. Усі правила OWASP CRS (SQL-ін'єкції, XSS, LFI, RFI тощо)
Include /usr/share/modsecurity-crs/rules/*.conf

# 4. Виключення false positives (додавати за потребою після тестування)
# SecRuleRemoveById 942100
```

Рис. 7.d – Вміст /etc/nginx/modsecurity/crs-setup.conf

e. Увімкнути ModSecurity у server block сайту. Додати у блок `server` файлу `/etc/nginx/sites-available/security-lab-xxx-yyy-zzz` (рис. 7.e):

```
server {
    listen 443 ssl;
    server_name security-lab-xxx-yyy-zzz.local;

    # ── ModSecurity WAF ──────────────────────────────────────
    # Увімкнути ModSecurity для цього server block
    modsecurity on;
    # Вказати файл конфігурації з правилами
    modsecurity_rules_file /etc/nginx/modsecurity/crs-setup.conf;

    # ... (SSL, root, limits, location — без змін) ...
}
```

Рис. 7.e – Додавання директив ModSecurity у server block nginx

f. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
sudo ls -la /var/log/nginx/modsec_audit.log
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo nginx -t
```

Зробити висновки за отриманими результатами.

g. Протестувати виявлення SQL-ін'єкцій (режим DetectionOnly). Запити не блокуються, але фіксуються у журналі аудиту:

```
# Тест 1: SQL-injection у параметрі URL (одинарна
лапка + OR)
curl -k -v \
    "https://security-lab-xxx-yyy-zzz.local/?id=1'
OR 1=1--"

# Тест 2: UNION-based SQL-injection
curl -k -s \
    "https://security-lab-xxx-yyy-zzz.local/?q=1
UNION SELECT 1,2,3,4--"

# Перевірити журнал аудиту
sudo tail -30 /var/log/nginx/modsec_audit.log \
    | python3 -m json.tool 2>/dev/null | head -50
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo tail -5 /var/log/nginx/modsec_audit.log
```

До застосування налаштування: Файл порожній — ModSecurity не активований або CRS не підключено.

Зробити висновки за отриманими результатами.

h. Протестувати виявлення XSS через POST-запит до контактної форми. Виконати:

```
# XSS через POST-параметр "name" (тег script)
curl -k -X POST \
    -d "name=<script>alert('XSS')</script>" \
    -d "email=attacker@evil.com" \
    -d "message=test message" \
    https://security-lab-xxx-yyy-
zzz.local/contact.html

# XSS через атрибут події (onerror)
curl -k -X POST \
    -d "name=<img src=x
onerror=alert(document.cookie)>" \
    -d "email=xss@test.com" \
    -d "message=benign" \
    https://security-lab-xxx-yyy-
zzz.local/contact.html
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo grep -i "XSS\|941\|script"
/var/log/nginx/modsec_audit.log | tail -5
```

Зробити висновки за отриманими результатами.

i. Перемкнути ModSecurity у режим On (активне блокування). У файлі `/etc/nginx/modsecurity/modsecurity.conf` замінити рядок `SecRuleEngine DetectionOnly` на `SecRuleEngine On` (рис. 7.i):

```
# Зміна у /etc/nginx/modsecurity/modsecurity.conf:

# Режим виявлення (було):
# SecRuleEngine DetectionOnly

# Режим активного блокування (стало):
SecRuleEngine On

# Тепер nginx повертатиме 403 Forbidden при виявленні атаки
```

Рис. 7.i – Перемикання SecRuleEngine у режим On (блокування)

```
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s -o /dev/null -w "%{http_code}"
"https://security-lab-xxx-yyy-zzz.local/?id=1' OR
1=1--"
```

До застосування налаштування: HTTP 200 OK — ModSecurity у режимі DetectionOnly або не активований.

Зробити висновки за отриманими результатами.

j. Налаштувати виключення для false positives — якщо певне правило блокує легітимні запити. Додати у кінець `/etc/nginx/modsecurity/crs-setup.conf` (рис. 7.j):

```
# Виключення false positives — додавати після Include правил CRS

# Вимкнення конкретного правила за ID (знаходити ID у журналі аудиту)
# SecRuleRemoveById 942100

# Виключення правила лише для конкретного URI (наприклад, форми)
SecRule REQUEST_URI "@beginsWith /contact.html"
    "id:9001,phase:1,pass,nolog,ctl:ruleRemoveById=941100"

# Виключення для конкретного параметру з конкретного IP (адмін)
# SecRule REMOTE_ADDR "@ipMatch 192.168.56.1"
#     "id:9002,phase:1,pass,nolog,ctl:ruleEngine=Off"
```

Рис. 7.j – Приклади виключень (false positive whitelisting)

8. Налаштувати додаткові заходи безпеки та підсумувати захист вебсервера.

a. Налаштувати кастомний формат журналювання для моніторингу підозрілої активності. Додати до блоку `http` файлу `/etc/nginx/nginx.conf` (рис. 8.a):

```
# Кастомний формат журналювання для аналізу безпеки
log_format security_log
    '$remote_addr - $remote_user [$time_local] '
    '"$request" $status $body_bytes_sent '
    '"$http_referer" "$http_user_agent" '
    'rt=$request_time ssl="$ssl_protocol/$ssl_cipher"';

# Формат включає:
# $remote_addr    — IP клієнта
# $time_local     — час запиту
# $request        — метод, URI, версія HTTP
# $status         — HTTP-код відповіді
# $body_bytes_sent— розмір тіла відповіді
# $http_user_agent— User-Agent (допомагає виявити бота)
# $request_time   — час обробки запиту
# $ssl_protocol   — TLS-протокол (TLSv1.2/TLSv1.3)
# $ssl_cipher     — використаний шифр
```

Рис. 8.a – Кастомний формат журналу security_log у блоці http

Застосувати формат у server block та перезавантажити nginx:

```
# Додати у server block:
# access_log /var/log/nginx/security-lab-access.log
security_log;
# error_log  /var/log/nginx/security-lab-error.log
warn;
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s https://security-lab-xxx-yyy-zzz.local >
/dev/null && sudo tail -3 /var/log/nginx/security-
lab-access.log
```

Зробити висновки за отриманими результатами.

b. Перевірити доступність чутливих файлів `.env` та `.git/` ДО налаштування захисту. Спостерігати наслідки місконфігурації — файли доступні ззовні:

```
curl -s -k \
    https://security-lab-xxx-yyy-zzz.local/.env
curl -s -k \
    https://security-lab-xxx-yyy-
zzz.local/.git/config
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s https://security-lab-xxx-yyy-
zzz.local/.env
```

Зробити висновки за отриманими результатами.

Налаштувати захист чутливих шляхів. Додати у server block (рис. 8.b):

```
# ── Блокування прихованих файлів та директорій (.git, .env тощо) ──
location ~ /\. {
    # Регулярний вираз ~/\./ виявляє будь-який шлях що містить /.
    # Блокує: /.env, /.git/config, /.htpasswd, /.ssh/id_rsa тощо
    deny all;
    access_log off;       # не записувати у access.log
    log_not_found off;    # не записувати у error.log
    return 404;           # повернути 404 (не 403 — не розкривати
існування)
}

# ── Блокування файлів з чутливими розширеннями ──────────────────
location ~*
\.(env|htpasswd|htaccess|bak|backup|sql|log|ini|conf|pem|key)$ {
    deny all;
    return 403;
}

# ── Явна заборона доступу до .git (додатковий захист) ──────────
location ^~ /.git/ {
    deny all;
    return 404;
}
```

Рис. 8.b – Захист чутливих шляхів у конфігурації nginx

c. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s -o /dev/null -w "%{http_code}"
https://security-lab-xxx-yyy-zzz.local/.env
```

Зробити висновки за отриманими результатами.

d. Налаштувати базову HTTP-автентифікацію для розділу `/admin`.

Встановити `apache2-utils` та створити файл паролів:

```
sudo apt install apache2-utils -y
# Створити файл паролів (-c: create new file)
sudo htpasswd -c /etc/nginx/.htpasswd admin
# Додати ще одного користувача (без -c, щоб не
перезаписати)
sudo htpasswd /etc/nginx/.htpasswd manager
# Встановити права: читати може nginx (www-data)
sudo chmod 640 /etc/nginx/.htpasswd
sudo chown root:www-data /etc/nginx/.htpasswd
sudo cat /etc/nginx/.htpasswd   # паролі у вигляді
хешів
```

Додати у server block блок location для /admin/ (рис. 8.d):

```
location /admin/ {
    # Базова HTTP-автентифікація (RFC 7617)
    # Браузер відображає діалог вводу логіну та паролю.
    # ВАЖЛИВО: використовувати ЛИШЕ з HTTPS! Без TLS паролі
    # передаються у відкритому вигляді (Base64, не шифрування).
```

Рис. 8.d-preview – Фрагмент конфігурації — базова автентифікація для /admin/

```
location /admin/ {
    auth_basic           "Restricted Area";
    auth_basic_user_file /etc/nginx/.htpasswd;

    # Суворий rate limit для захисту від brute force на пароль
    limit_req zone=auth burst=3 nodelay;

    try_files $uri $uri/ =404;
}
```

Рис. 8.d – Конфігурація auth_basic для /admin/ у server block

```
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s -o /dev/null -w "%{http_code}"
https://security-lab-xxx-yyy-zzz.local/admin/
```

Зробити висновки за отриманими результатами.

e. Заборонити виконання скриптів у каталозі `/uploads`. Додати у server block (рис. 8.e):

```
location /uploads/ {
    # Захист від завантаження та виконання веб-шелів.
    # Навіть якщо зловмисник завантажить shell.php — він не виконається.

    # Заблокувати запити до файлів з виконуваними розширеннями
    location ~* \.(php|php3|php4|php5|php7|php8|phtml
                   |pl|py|rb|sh|cgi|asp|aspx|jsp|cfm)$ {
        deny all;
        return 403;
    }

    # Обслуговувати лише наявні статичні файли
    try_files $uri =404;
}
```

Рис. 8.e – Захист від виконання скриптів у /uploads/

f. Перевірити синтаксис та перезавантажити nginx, виконавши:

```
sudo nginx -t && sudo nginx -s reload
```

Перевірити застосовану конфігурацію, виконавши команду:

```
curl -k -s -o /dev/null -w "%{http_code}"
https://security-lab-xxx-yyy-
zzz.local/uploads/shell.php
```

Зробити висновки за отриманими результатами.

h. Зібрати підсумкову захищену конфігурацію вебсервера. Переглянути фінальний вигляд файлу `/etc/nginx/sites-available/security-lab-xxx-yyy-zzz` з усіма заходами безпеки, застосованими у завданнях 2–8 (рис. 8.h). Конфігурація є результатом послідовного накопичення всіх захисних механізмів:

```
server {
    listen 80;
    server_name security-lab-xxx-yyy-zzz.local security-lab-xxx-yyy-zzz.pp.ua;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name security-lab-xxx-yyy-zzz.local security-lab-xxx-yyy-zzz.pp.ua;

    ssl_certificate     /etc/letsencrypt/live/security-lab-xxx-yyy-
zzz.pp.ua/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/security-lab-xxx-yyy-
zzz.pp.ua/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    autoindex off;
    include /etc/nginx/snippets/security-headers.conf;
    if ($request_method !~ ^(GET|HEAD|POST)$) { return 405; }

    client_max_body_size 1m;
    client_body_timeout 10s;
    client_header_timeout 10s;
    send_timeout 10s;
    keepalive_timeout 30s;
    limit_req  zone=general burst=20 nodelay;
    limit_conn conn_limit 20;

    add_header Strict-Transport-Security "max-age=63072000; includeSubDomains"
always;

    modsecurity on;
    modsecurity_rules_file /etc/nginx/modsecurity/crs-setup.conf;

    access_log /var/log/nginx/security-lab-access.log security_log;
    error_log  /var/log/nginx/security-lab-error.log warn;

    root  /var/www/security-lab-xxx-yyy-zzz;
    index index.html;

    location ~ /\.          { deny all; access_log off; log_not_found off; return
404; }
    location ^~ /.git/      { deny all; return 404; }
    location ~* \.(env|htpasswd|htaccess|bak|backup|sql|log|ini|conf|pem|key)$ {
deny all; return 403; }

    location /admin/ {
        auth_basic "Restricted Area";
        auth_basic_user_file /etc/nginx/.htpasswd;
        limit_req zone=auth burst=3 nodelay;
        try_files $uri $uri/ =404;
    }

    location /uploads/ {
        location ~* \.(php|php[3-8]|phtml|pl|py|rb|sh|cgi|asp|aspx|jsp|cfm)$ {
deny all; return 403; }
        try_files $uri =404;
    }

    location / { try_files $uri $uri/ =404; }
}
```

Рис. 8.h – Підсумкова захищена конфігурація nginx з усіма заходами безпеки

i. Виконати фінальну перевірку синтаксису конфігурації та перезавантажити nginx:

```
sudo nginx -t
sudo nginx -s reload
sudo systemctl status nginx --no-pager
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo nginx -t
```

Зробити висновки за отриманими результатами.

j. Провести фінальну перевірку всіх налаштованих механізмів безпеки. Виконати серію перевірок та звірити результати з таблицею очікуваних відповідей:

```
# ── Перевірка 1: server_tokens приховано ──────────────────────────
curl -I -k \
    https://security-lab-xxx-yyy-zzz.local/ 2>&1 | grep -i server
# Очікується: Server: nginx  (без версії)
# ── Перевірка 2: security headers присутні ────────────────────────
curl -I -k \
    https://security-lab-xxx-yyy-zzz.local/ 2>&1 \
    | grep -iE "X-Content|X-Frame|X-XSS|Content-Security|Strict-
Transport"
# Очікується: усі 5 заголовків безпеки + HSTS
# ── Перевірка 3: небезпечні методи заблоковані ────────────────────
curl -k -s -o /dev/null -w "DELETE: %{http_code}\n" -X DELETE \
    https://security-lab-xxx-yyy-zzz.local/
# Очікується: DELETE: 405
# ── Перевірка 4: .env недоступний ─────────────────────────────────
curl -k -s -o /dev/null -w ".env: %{http_code}\n" \
    https://security-lab-xxx-yyy-zzz.local/.env
# Очікується: .env: 404
# ── Перевірка 5: /admin/ вимагає автентифікацію ───────────────────
curl -k -s -o /dev/null -w "admin: %{http_code}\n" \
    https://security-lab-xxx-yyy-zzz.local/admin/
# Очікується: admin: 401
# ── Перевірка 6: ModSecurity блокує SQL-ін'єкцію ──────────────────
curl -k -s -o /dev/null -w "sqli: %{http_code}\n" \
    "https://security-lab-xxx-yyy-zzz.local/?id=1' OR 1=1--"
# Очікується: sqli: 403 (SecRuleEngine On)
# ── Перевірка 7: /uploads/ блокує PHP-файли ───────────────────────
curl -k -s -o /dev/null -w "shell.php: %{http_code}\n" \
    https://security-lab-xxx-yyy-zzz.local/uploads/shell.php
# Очікується: shell.php: 403
# ── Перевірка 8: TLS-версія ───────────────────────────────────────
openssl s_client -connect 127.0.0.1:443 \
    -servername security-lab-xxx-yyy-zzz.local </dev/null 2>&1 \
    | grep "Protocol"
# Очікується: Protocol: TLSv1.2 або TLSv1.3
```

Рис. 8.j – Фінальний скрипт перевірки всіх захисних механізмів

Перевірити застосовану конфігурацію, запустивши перевірки.

Зробити висновки за отриманими результатами.

k. Переглянути журнали подій безпеки та переконатись у збереженні інформації про всі підозрілі запити у ході виконання перевірок:

```
# Переглянути кастомний журнал доступу
sudo tail -20 /var/log/nginx/security-lab-access.log

# Переглянути журнал ModSecurity (останні події)
sudo tail -20 /var/log/nginx/modsec_audit.log
2>/dev/null \
    | python3 -c "import sys,json;
[print(json.dumps(json.loads(l), indent=2,
ensure_ascii=False)) for l in sys.stdin if
l.strip()]" 2>/dev/null | head -40

# Переглянути журнал Fail2Ban
sudo tail -20 /var/log/fail2ban.log

# Переглянути журнал помилок nginx
sudo tail -10 /var/log/nginx/security-lab-error.log
```

Перевірити застосовану конфігурацію, виконавши команду:

```
sudo tail -5 /var/log/nginx/security-lab-access.log
```

Зробити висновки за отриманими результатами.

l. Зупинити VM після завершення роботи, виконавши:

```
exit
vagrant halt
```

Контрольні запитання

1. Яку роль відіграє директива server_tokens off у забезпеченні безпеки вебсервера та яку інформацію вона приховує від потенційного зловмисника?
2. У чому полягає призначення HTTP-заголовків безпеки X-Content-Type-Options, X-Frame-Options, X-XSS-Protection та Content-Security-Policy? Яким атакам запобігає кожен з них?
3. Як відбувається захист від атак типу Slowloris та slow HTTP за допомогою таймаутів nginx? Які директиви відповідають за обмеження часу читання заголовків та тіла запиту?
4. У чому полягає різниця між rate limiting (limit_req_zone) та connection limiting (limit_conn_zone)? Яким чином параметри burst та nodelay впливають на поведінку rate limiting?
5. Яку роль відіграє заголовок Strict-Transport-Security (HSTS) та чому значення max-age рекомендують встановлювати не менше двох років для виробничого середовища?
6. Яким чином DNS-01 challenge відрізняється від HTTP-01 challenge у Certbot та в яких сценаріях використання DNS-01 є єдиним можливим методом отримання сертифіката Let's Encrypt?
7. Як відбувається автоматичне оновлення сертифікатів Let's Encrypt через systemd-таймер? Яка різниця між перевіркою renew та реальним оновленням сертифіката?
8. У чому полягає принцип роботи Fail2Ban? Яким чином параметри bantime, findtime та maxretry у jail-конфігурації разом визначають чутливість системи захисту?
9. Які переваги надає WAF (ModSecurity) порівняно з мережевими брандмауерами та rate limiting? Які типи вебатак здатен виявляти та блокувати OWASP Core Rule Set?
10. Яким чином режим SecRuleEngine DetectionOnly допомагає при початковому розгортанні ModSecurity та чому не рекомендується одразу переходити до режиму On?
11. У чому полягає небезпека потрапляння файлів .env та .htpasswd до публічного git-репозиторію та яким чином захист на рівні nginx частково компенсує цю помилку розробника?
12. Яким чином базова HTTP-автентифікація (auth_basic) взаємодіє з SSL/TLS і чому її використання без HTTPS є критичною вразливістю?