HoneyScope — Honeypot de baja interacción con dashboard

> **Asignatura (ficción):** Seguridad de Sistemas y Redes
> **Título del proyecto (para el repositorio):** `honeyscope`
> **Dedicación estimada:**
>
> - **Nivel MVP:** 20–30 horas
> - **Nivel Pro:** +25–35 horas adicionales (total 45–65 h)

---

1) Resumen del proyecto

Idea: Construir un **honeypot de baja interacción** que simule un servicio expuesto a Internet (a elegir: **SSH** *o* **HTTP**) para **capturar y registrar intentos de intrusión** (IPs, credenciales/payloads, timestamps, etc.).
El sistema incluirá un **dashboard web** con métricas y tablas para visualizar la actividad recogida.

Qué valida de tus habilidades (learning outcomes):

- **Linux & Redes:** sockets/puertos, firewall básico, despliegue seguro en VM/contenerización.
- **Ingeniería del Software:** diseño modular (servicio → ingesta → almacenamiento → API → UI), testing, CI/CD, documentación.
- **Ciberseguridad práctica (blue team):** registro de IOCs, análisis básico de TTPs, buenas prácticas de exposición segura y ética.

> **Importante:** Honeypot de **baja interacción**. No se ejecutan binarios ni comandos del atacante. Solo **simulación** y **logging**.

---

2) Alcance y criterios de finalización

2.1 Nivel **MVP** (proyecto funcional y "portfolio-ready")

Se considera **COMPLETADO** cuando cumpla **todo** lo siguiente:

1. **Servicio honeypot (elige uno):**
   - **SSH falso** (recomendado si eliges Node/TS: librería `ssh2` en modo servidor)
     o **HTTP falso** (Express/Fastify; rutas que simulan login o endpoints tentadores).
   - Puerto configurable (por defecto **22** si SSH, **80** si HTTP).
   - Registra **por evento**:
     - `timestamp` (UTC), `src_ip`, `src_port`
     - **SSH**: `username`, `password` (si hay intento), `client_id`/`kex` (si es fácil de obtener).
     - **HTTP**: método, path, headers mínimos, body (limitado/tamizado).
   - **No** ejecutar nada del atacante; **no** abrir shell real.
2. **Almacenamiento estructurado**
   - Persistencia en **SQLite** (fichero local) con esquema claro e índices mínimos.

- Rotación simple o tamaño máximo del fichero (documentada).

3. **API interna (REST)**
   - Endpoints **read-only** para el dashboard (ej.: `/api/events?limit=...`, `/api/stats/summary`).
   - Paginación en `/api/events`.
   - **Autorización** con token de administración para la API (header `Authorization: Bearer ...`).
4. **Dashboard web (Next.js)**
   - **Gráficas**: intentos por día, top IPs origen, top usernames/paths.
   - **Tabla** con los últimos N eventos (búsqueda y ordenación básica).
   - Configuración de `BASE_API_URL` por variables de entorno.
5. **Contenerización y ejecución**
   - **Dockerfile** y **docker-compose.yml** que levanten:
     - `honeypot` (servicio)
     - `api` (si separas proceso) o integrado
     - `dashboard`
   - Una orden única de demo: `docker compose up -d` (o `make demo`).
6. **Configuración por entorno**
   - `.env.example` con variables:
     - `HNY_SERVICE` = `ssh` | `http`
     - `HNY_PORT` (p.ej. 22/80/8080)
     - `HNY_DB_PATH` (p.ej. `./data/events.db`)
     - `HNY_ADMIN_TOKEN` (token para API)
     - `DASHBOARD_BASE_URL`, `BASE_API_URL`
7. **Seguridad mínima y ética**
   - El proceso corre **sin privilegios** (usuario no root en contenedor).
   - Exposición de **solo** el puerto honeypot.
   - Aviso en README sobre **uso ético** y **riesgos**.
   - No almacenar datos personales más allá de metadatos técnicos necesarios.
8. **Calidad & DevEx**
   - **README** completo (ver sección “Entrega esperada”).
   - **CI**: GitHub Actions con `lint` + `test` + `build`.
   - **Tests básicos**:
     - Unit (parser/normalizador de eventos).
     - Integración (simular 1–2 ataques de ejemplo y verificar que se guardan y aparecen en la API).

> **Fin del MVP:** Con estos puntos, puedes **cerrar el proyecto** con tranquilidad y enseñarlo en entrevistas.

2.2 Nivel **Pro** (para un 10/10 y “wow factor”)

Añade **al menos 5** de las siguientes funcionalidades extra:

- **Multi-servicio**: SSH **y** HTTP en paralelo (o añadir Telnet/FTP falsos como bonus).
- **GeoIP**: enriquecer eventos con país/ASN (guárdalo en campos separados).
- **Alertas**: webhook a Discord/Slack ante umbrales (p. ej. >100 intentos/min).
- **Prometheus**: endpoint `/metrics` con counters y gauges básicos.
- **Mapa en dashboard**: ataques por país.
- **Filtros avanzados** en UI: por IP, rango de fechas, usuario.
- **Exportación**: endpoints para CSV/NDJSON; integración opcional con ELK (enviar a Logstash).
- **Rate-limit & WAF light** en los endpoints API y dashboard.
- **TLS** para dashboard/API tras reverse proxy (nginx/Traefik) + headers de seguridad (CSP/HSTS).
- **Despliegue reproductible**: Terraform + Ansible en una VPS (Hetzner/OVH/DO).
- **Logs firmados** o hash chain (integridad básica de registros).
- **Panel “acciones rápidas”**: bloquear IP en firewall (simulado o real si integras nftables).

---

3) Arquitectura de referencia

> Puedes **seguir esta arquitectura o justificar cambios**. Mantén separación clara entre **captura** → **almacenamiento** → **API** → **UI**.

```
flowchart LR
  A[Internet] -->|conexiones reales| B[Servicio Honeypot
SSH o HTTP]
  B --> C[Normalizador/
Collector]
  C --> D[(SQLite)]
  D --> E[API REST
Read-only]
  E --> F[Dashboard
Next.js]
```

- **Servicio Honeypot:** escucha en el puerto, simula respuestas mínimas y entrega eventos al **Collector**.
- **Collector:** sanitiza y guarda en **SQLite**.
- **API:** ofrece estadísticas y eventos (paginados).
- **Dashboard:** consume la API y muestra gráficas/tablas.

Esquema de datos (SQLite, referencia)

```
CREATE TABLE IF NOT EXISTS events (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  ts_utc TEXT NOT NULL,          -- ISO 8601
  src_ip TEXT NOT NULL,
  src_port INTEGER,
  service TEXT NOT NULL,         -- 'ssh' | 'http'
  username TEXT,                 -- ssh
```

```
  password TEXT,                -- ssh (si procede, opcional)
  http_method TEXT,             -- http
  http_path TEXT,               -- http
  http_status INTEGER,          -- http (si simulas respuesta)
  user_agent TEXT,              -- http
  raw JSON,                     -- campos extra según servicio (limitado)
  country TEXT,                 -- pro (GeoIP)
  asn TEXT                      -- pro (GeoIP)
);

CREATE INDEX IF NOT EXISTS idx_events_ts ON events(ts_utc);
CREATE INDEX IF NOT EXISTS idx_events_ip ON events(src_ip);
CREATE INDEX IF NOT EXISTS idx_events_service ON events(service);
```

Endpoints REST (obligatorios)

- `GET /api/stats/summary` → `{ totalEvents, byDay[], topIPs[], topUsernames[], topPaths[] }`
- `GET /api/events?limit=50&offset=0&service=ssh|http&ip=...&from=...&to=...`
- **Auth**: header `Authorization: Bearer <HNY_ADMIN_TOKEN>` (solo lectura).

---

4) Requisitos de implementación

4.1 Stack recomendado (orientativo, puedes proponer variaciones)

- **Backend / Honeypot & API:** Node.js + TypeScript
  - SSH servidor: `ssh2` (modo server).
  - HTTP servidor: Fastify/Express.
  - DB: SQLite (con Prisma o `better-sqlite3`).
- **Dashboard:** Next.js (App Router), charts (Recharts).
- **Contenedores:** Docker; `docker-compose` para orquestar.
- **Scripting opcional:** Makefile con atajos (`make dev`, `make demo`, `make test`).
- **CI/CD:** GitHub Actions (jobs de lint/test/build y publicar imagen en GHCR opcional).

> **Alternativa** (válida si lo prefieres): Go (net/http, sqlite), API integrada y dashboard aparte.

4.2 Estructura del repositorio (sugerida)

```
honeyscope/
├─ apps/
│  ├─ honeypot/            # servicio (ssh|http) + collector
│  ├─ api/                 # API REST read-only
│  └─ dashboard/           # Next.js
├─ packages/
│  ├─ db/                  # schema y cliente sqlite
│  └─ common/              # tipos compartidos
├─ docker/
│  └─ Dockerfile.*         # imágenes
├─ docker-compose.yml
```

```
├─ .env.example
├─ Makefile
└─ README.md
```

4.3 Variables de entorno (mínimas)

```
HNY_SERVICE=ssh        # o http
HNY_PORT=22            # 22 si ssh, 80/8080 si http
HNY_DB_PATH=./data/events.db
HNY_ADMIN_TOKEN=change-me
BASE_API_URL=http://api:3000
DASHBOARD_BASE_URL=http://localhost:3001
```

---

5) Despliegue y entorno de ejecución

5.1 Desarrollo local (obligatorio)

- Ejecución con `docker compose up -d` que levante honeypot, api y dashboard.
- Dashboard accesible en `http://localhost:3001`.
- Simulaciones de ataque locales (`curl` a HTTP o cliente SSH apuntando a tu puerto) para generar eventos de ejemplo.

5.2 Despliegue en VPS (recomendado)

- VM Linux dedicada, usuario sin privilegios, sistema actualizado.
- Abrir **solo** el puerto del honeypot al exterior (p. ej., 22/80).
- Acceso de administración **vía WireGuard** (opcional pero recomendado).
- Correr los contenedores con usuario **no root** y capacidades limitadas.
- Guardar la DB en volumen persistente; backup básico documentado.

> **No** desplegar en máquinas con datos personales. **No** mezclar con tu servidor principal.

---

6) Seguridad, privacidad y ética

- Honeypot de **baja interacción**: **no** ejecutes nada que venga del atacante.
- **No contraataques** ni escaneos de vuelta. Fines únicamente didácticos.
- Almacena **solo metadatos necesarios**. Indica retención y purga en README (p. ej., 30–90 días).
- Documenta claramente que la IP y datos recogidos provienen de **intentos de acceso a un servicio simulado**.

---

7) Testing (mínimo exigido)

- **Unit tests**: normalización de eventos y validación de entradas.
- **Integración**: levantar el servicio en local y generar 2–3 eventos sintéticos; comprobar que la API los devuelve.

- **E2E (opcional)**: flujo completo con `docker compose` en CI.

---

8) Entrega esperada (para evaluación/portfolio)

Tu **README del repo** debe incluir:

1. **Descripción** del proyecto y motivación.
2. **Arquitectura** (incluye diagrama Mermaid similar al de este enunciado).
3. **Guía de ejecución**:
   - Requisitos.
   - Variables de entorno (proveer `.env.example`).
   - Comandos: `docker compose up -d`, `make demo`.
4. **Demo**:
   - Cómo generar eventos de prueba (curl/ssh).
   - Capturas o gifs del dashboard (si corresponde).
5. **API**: endpoints documentados (ruta, params, respuesta ejemplo).
6. **Esquema de datos**: tablas y campos importantes.
7. **Seguridad**: medidas adoptadas y límites del sistema.
8. **Tests**: cómo correrlos y qué cubren.
9. **Roadmap**: lista de mejoras futuras (incluye ítems del nivel Pro).
10. **Licencia** y **créditos** (si usas assets/paquetes relevantes).

---

9) Criterios de evaluación (rubrica resumida)

Criterio	MVP	Pro
Funciona end-to-end	✅	✅
Calidad de código y estructura	✅	✅
Seguridad mínima y aislamiento	✅	✅
Dashboard útil y claro	✅	✅
Documentación y reproducibilidad	✅	✅
Multi-servicio / features avanzadas		✅
Observabilidad (metrics/alertas)		✅
Despliegue infra (Terraform/Ansible)		✅

---

10) Plan de trabajo sugerido (sprints cortos)

- **Sprint 0 (2–3 h):** bootstrap repo, `.env.example`, estructura, CI de lint/build.
- **Sprint 1 (5–8 h):** servicio honeypot (HTTP o SSH), eventos a stdout y fichero.
- **Sprint 2 (4–6 h):** SQLite + collector + API `/api/events`, `/api/stats/summary`.
- **Sprint 3 (4–6 h):** dashboard con 2 gráficas + tabla con paginación.
- **Sprint 4 (3–5 h):** Dockerfiles, `docker-compose`, hardening básico, README final, tests mínimos.

(Para nivel Pro, añade 2–3 sprints de 5–6 h cada uno.)

---

11) Decisiones explícitas del alumno

Debes **elegir y justificar** (en tu README):

- Servicio del MVP: **SSH** *o* **HTTP**.
  - Recomendación: si dominas Node/TS, **SSH** con `ssh2` produce eventos interesantes; **HTTP** es aún más sencillo y válido.
- Implementación monolito vs. servicios separados (honesto justificar simplicidad).
- Paquetes/librerías elegidos (y por qué).
- Parámetros de retención de logs y límites de tamaño.

📌 Recordatorio final (punto de cierre)

El proyecto se considera **"COMPLETADO (MVP)"** cuando:

- Capturas eventos reales o simulados del servicio elegido.
- Se almacenan en SQLite con el esquema definido.
- La API devuelve estadísticas y eventos paginados con **auth por token**.
- El dashboard muestra **al menos**: intentos por día, top IPs, top usernames/paths y la tabla de eventos.
- Existe contenedorización reproducible y un README que permita a un tercero levantarlo sin dudas.
- Hay **tests básicos** y **CI** que pasan.

A partir de ahí, cualquier mejora del **Nivel Pro** suma puntos para entrevistas y demuestra ambición técnica.

¡Éxitos y a cazar paquetes (de forma ética)! 🦋