# 🕵️ Honey-Scan: Tiefenanalyse des Codes

## 1. Executive Summary

Das Tool **Honey-Scan** ist ein spezialisiertes Sicherheitswerkzeug, das entwickelt wurde, um Netzwerke oder Hosts zu scannen und zu identifizieren, ob es sich dabei um **Honeypots** (Täuschungssysteme) handelt. Es arbeitet proaktiv (Scannen) statt reaktiv.

Der Kernprozess basiert auf **Fingerprinting**: Das Tool sendet spezifische Payloads an Zielsysteme und analysiert die Antworten (Banners, Latenzzeiten, Fehlercodes) auf Signaturen, die typisch für bekannte Honeypots wie *Cowrie*, *Dionaea* oder *Kippo* sind.

## 2. Architektur & Hauptfunktionen

### ⚙️ Core Engine (Der Motor)

Das Herzstück ist eine asynchrone oder Multi-Threaded Loop, die IPs aus einer Warteschlange abarbeitet.

- **main.py (Entry Point):**
  - **Input:** CLI-Argumente (Ziel-IP, Port-Range, Output-Format).
  - **Trigger:** Startet den Controller.
- **scanner.py (Die Sonde):**
  - Erstellt TCP/UDP Sockets.
  - Führt den "Handshake" durch.
  - **Wichtig:** Sendet oft absichtlich "falschen" Traffic, um Fehlermeldungen zu provozieren, die den Honeypot verraten.
- **signatures.py (Die Datenbank):**
  - Enthält Dictionarys oder JSON-Strukturen mit RegEx-Mustern.
  - Beispiel: if "svr04" in banner: return "Dionaea"

### 🔄 Datenfluss & API Beziehungen

1. **Ingest (Datenaufnahme):**
   - Der User gibt eine IP oder ein Subnetz ein.
   - Optional: Import von Listen (z.B. von Shodan oder lokaler TXT).
2. **Processing (Verarbeitung):**
   - Die IP wird in ein Target Object umgewandelt.
   - Der Scanner iteriert über definierte Ports (22, 23, 80, 443, etc.).
3. **Detection Logic (Erkennung):**
   - Datenstrom (Response) wird gegen signatures.py gematcht.
   - **Scoring:** Ein "Honeyscore" wird berechnet (Wahrscheinlichkeit 0.0 - 1.0).

4. **Egress (Ausgabe):**
    - Strukturierte Daten (JSON) werden erstellt.
    - Archivierung lokal oder Weitergabe an STDOUT.

# 3. Trigger & Prozessablauf

- **Trigger:** Manuelle Ausführung durch den User (python honey-scan.py -t 192.168.1.1).
- **Datenentstehung:** Entsteht im Moment des socket.recv(). Die rohen Bytes sind die primäre Datenquelle.
- **Verarbeitung:**
    - Decoding (Bytes -> UTF-8).
    - Normalisierung (Lowercasing, Whitespace removal).
    - Matching (Regex).
- **Verwurf:** Wenn kein Port offen ist oder Timeouts auftreten, wird das Ziel verworfen bzw. als "Clean" markiert.

# 4. Visuelle Analyse (Mermaid Diagramme)

## A. High-Level System Overview

Dieses Diagramm zeigt den Weg vom User bis zum fertigen Log-File.

```
graph TD
    User([👤 User / CLI]) -->|Start Command| ArgParse[Arguments Parser]
    ArgParse -->|Config Object| Controller{⚙️ Main Controller}

    subgraph "Core Logic"
        Controller -->|Dispatch Targets| WorkerPool[Thread/Async Pool]
        WorkerPool -->|1. Connect| SocketHandler[Socket Handler]
        SocketHandler <-->|2. Handshake/Payload| Target((🎯 Target IP))
        SocketHandler -->|3. Raw Data| Analyzer[🔍 Signature Analyzer]

        Analyzer -.->|Reference| SigDB[(📂 Signatures DB)]
    end

    Analyzer -->|Result Object| Aggregator[Result Aggregator]
    Aggregator -->|Format| JSON_Export[📄 JSON Output]
    Aggregator -->|Format| Console[🖥️ Console Output]
```

## B. Detailprozess: Die Erkennungslogik (The Brain)

Hier sehen wir, wie die Entscheidung "Honeypot: Ja/Nein" getroffen wird.

```
flowchart LR
```

```
    Start([Raw Response Bytes]) --> Decode{Decode UTF-8?}
    Decode -->|Success| StringProcess[Normalize String]
    Decode -->|Fail| HexAnalyze[Analyze Hex Patterns]

    StringProcess --> RegexLoop{Iterate RegEx Patterns}
    HexAnalyze --> HexLoop{Iterate Hex Patterns}

    RegexLoop -->|Match Found| ScoreUp[⚠️ Increase Honeyscore]
    HexLoop -->|Match Found| ScoreUp

    RegexLoop -->|No Match| NextPattern

    ScoreUp --> Threshold{Score > Threshold?}
    Threshold -->|Yes| VerdictBad[🚨 Verdict: HONEYPOT]
    Threshold -->|No| VerdictGood[✅ Verdict: LEGIT SYSTEM]
```

## C. Sequenzdiagramm: API/Network Interaction

Der zeitliche Ablauf einer einzelnen Überprüfung.

```
sequenceDiagram
    participant Ctrl as MainController
    participant Scan as ScannerModule
    participant Target as TargetHost
    participant DB as SignatureDatabase

    Ctrl->>Scan: scan_target(IP: 10.0.0.5)
    activate Scan
    Scan->>Target: TCP SYN (Port 22)
    Target-->>Scan: SYN-ACK
    Scan->>Target: Send Fake SSH Version
    Target-->>Scan: Server Protocol Mismatch (Raw Data)

    Scan->>DB: get_signatures("SSH")
    DB-->>Scan: return [Patterns...]

    loop Pattern Matching
        Scan->>Scan: match(RawData, Pattern)
    end

    Scan-->>Ctrl: return Result(IsHoneypot=True, Type="Kippo")
    deactivate Scan
```