Of course. Let's transition from the "what" to the "how." This is the detailed, low-level implementation plan for **Project Chimera**.

This document will serve as a technical blueprint for the engineering and data science teams tasked with building this next-generation fraud detection system. It will cover the microservice architecture, API contracts, data models, technology stack, and real-time data flows.

Project Chimera: Low-Level Design (LLD)

1. Core Architectural Principles

- **Microservices Architecture:** The system will be composed of small, independent, and specialized services. This allows for independent scaling, deployment, and technology choices for each component.
- **Asynchronous Communication (where possible):** For non-blocking operations like updating a behavioral profile, we will use a message queue (e.g., Kafka, RabbitMQ) to decouple services and improve resilience.
- **Real-Time Synchronous Communication (for scoring):** The core scoring path will be synchronous to provide an immediate response within a strict sub-100ms Service Level Agreement (SLA).
- **Stateless Services:** All services will be designed to be stateless, with state managed in external databases (Redis, Neo4j, PostgreSQL). This is critical for horizontal scalability.
- **Infrastructure as Code (IaC):** The entire infrastructure will be defined using tools like Terraform or CloudFormation for repeatable, version-controlled deployments.

2. Detailed Microservice Breakdown

Project Chimera will be composed of six core microservices, fronted by an API Gateway.

A. API Gateway

- **Role:** Single, secure entry point for all incoming traffic.
- **Technology:** Kong, AWS API Gateway, or Apigee.
- **Responsibilities:**
  - Authenticate incoming requests from merchants (API Key/JWT validation).
  - Handle request/response transformations.
  - Route requests to the `Orchestration Service`.
  - Implement rate limiting and basic security policies.

B. Orchestration Service (`sentinel-core`)

- **Role:** The real-time brain of the system. It manages the fraud scoring workflow.
- **Technology:** Go or Rust (for maximum performance and low-latency concurrency) or a non-blocking Java/Kotlin framework (e.g., Spring WebFlux, Vert.x).
- **API Endpoint:** `POST /v1/score`
- **Internal Logic:**
  i. Receives the transaction payload from the API Gateway.
  ii. Fires off **parallel, synchronous** API calls to the Cognito, Praxis, and Nexus services.
  iii. Waits for all responses, with a strict internal timeout (e.g., 80ms).
  iv. Aggregates the sub-scores from each service.
  v. Applies a **weighted scoring model** (a small, fast internal ML model or a configurable ruleset) to calculate the final `chimeraScore`.
  vi. Based on the final score and reason codes, makes a `decision` (`ALLOW`, `REVIEW`, `DENY`).
  vii. If the decision is `REVIEW`, it makes an asynchronous call to the `Trickster Service` to pre-generate a challenge URL.
  viii. Returns the final JSON response to the API Gateway.

C. Cognito Service (`identity-analyzer`)

- **Role:** Analyzes all identity-related attributes.
- **Technology:** Python (leveraging TensorFlow/PyTorch, OpenCV, and data science libraries).
- **API Endpoint:** `POST /v1/analyze/identity`
- **Internal Logic:**
  - Checks email/phone reputation via third-party APIs (e.g., Ekata) or internal models.
  - If document/liveness data is present, it runs it through a pre-trained **CNN/Vision Transformer model** for forgery and deepfake detection.
  - Queries the `Nexus Service` to check for identity element linkage to known fraud.
  - Returns a sub-score and reason codes (e.g., `EMAIL_HIGH_RISK`, `DEEPFAKE_DETECTED`).

D. Praxis Service (`behavior-analyzer`)

- **Role:** Analyzes user behavioral biometrics.
- **Technology:** Python (with Scikit-learn, Keras/TensorFlow).
- **API Endpoint:** `POST /v1/analyze/behavior`
- **Internal Logic:**
  - Retrieves the user's historical behavioral baseline from a **Redis cache** (for speed) or a primary DB.
  - Feeds the current session's event stream (keystroke timings, mouse dynamics) into an

LSTM Autoencoder or Isolation Forest model.
    - Calculates an anomaly score.
    - Detects patterns indicative of bots or AI-mimicry (e.g., unnaturally perfect or jittery movements).
    - Asynchronously updates the user's baseline in the database.
    - Returns an anomaly sub-score and reason codes (e.g., `BEHAVIOR_ANOMALOUS`, `BOT_LIKE_INTERACTION`).

E. Nexus Service (`graph-investigator`)

- **Role:** Uncovers hidden connections and coordinated fraud rings.
- **Technology:** A Python/Java service fronting a **Graph Database (Neo4j or Amazon Neptune)**.
- **API Endpoint:** `POST /v1/query/links`
- **Internal Logic:**
    - Takes identifiers from the request (Device ID, IP, User ID, Address Hash).
    - Constructs and executes a **Cypher/Gremlin query** against the graph database.
    - Searches for patterns like:
        - High fan-out (one device linked to many users).
        - Proximity to known fraudulent nodes.
        - Unusual ring structures (money cycling).
    - Returns a network risk sub-score and reason codes (e.g., `LINKED_TO_FRAUD_RING`, `SYNTHETIC_IDENTITY_PATTERN`).

F. Trickster Service (`challenge-generator`)

- **Role:** Generates and verifies dynamic, AI-resistant challenges.
- **Technology:** A simple, stateless service (e.g., Python with Flask, or Node.js).
- **API Endpoints:**
    - `POST /v1/challenge/generate`: Receives context (e.g., `risk_factor: "high-risk IP"`) and returns a JSON object containing the challenge details (e.g., image URLs, question text) and a signed JWT containing the correct answer and an expiry.
    - `POST /v1/challenge/verify`: Receives the user's answer and the JWT. Verifies the JWT signature and expiry, then checks the answer. Returns `{ "status": "passed" | "failed" }`.

3. Data Models & API Contracts

A. Main Scoring Request: `POST /v1/score`

```
// Request Body from Merchant
{
  "transactionId": "txn_12345",
  "user": {
    "id": "cust_abc",
    "email": "test@example.com",
    "phone": "15551234567",
    "billingAddress": { /* ... */ }
  },
  "transaction": {
    "amount": 99.99,
    "currency": "USD",
    "paymentMethod": {
      "type": "card",
      "bin": "424242"
    }
  },
  "device": {
    "fingerprintId": "fp_xyz789",
    "ipAddress": "123.45.67.89"
  },
  "session": {
    "behavioralData": [ /* Array of keystroke/mouse events */ ],
    "livenessCheck": {
      "video_b64": "..." // Optional
    }
  }
}
```

B. Main Scoring Response: `200 OK`

```
// Response Body to Merchant
{
  "chimeraId": "chi_abcdef123",
  "decision": "REVIEW", // ALLOW, REVIEW, DENY
  "chimeraScore": 85, // Overall risk score 0-100
  "reasonCodes": [
    {
      "code": "DEVICE_ID_HAS_HIGH_FANOUT",
      "description": "This device has been used with multiple other customer accoun
      "contribution": 45
    },
    {
      "code": "BEHAVIOR_ANOMALOUS",
      "description": "User interaction patterns deviate significantly from historic
      "contribution": 30
    }
  ],
  "challenge": { // Only present if decision is REVIEW
```

```
    "type": "dynamic_image_drag",
    "challengeUrl": "https://chimera.api/challenge/show/jwt.token.here"
  }
}
```

C. Graph Database Schema (Simplified)

- **Nodes:** `(:User)`, `(:Device)`, `(:PaymentMethod)`, `(:IPAddress)`, `(:Address)`
- **Edges:**
  - `[:LOGGED_IN_FROM]` (User -> IP)
  - `[:USED_DEVICE]` (User -> Device)
  - `[:HAS_PAYMENT_METHOD]` (User -> PaymentMethod)
  - `[:HAS_ADDRESS]` (User -> Address)
  - `[:INITIATED_FROM]` (PaymentMethod -> Device)

4. Real-Time Data Flow (Sequence Diagram)

```
sequenceDiagram
    participant Merchant as Merchant Backend
    participant Gateway as Chimera API Gateway
    participant Sentinel as Orchestration Svc
    participant Cognito as Cognito Svc
    participant Praxis as Praxis Svc
    participant Nexus as Nexus Svc

    Merchant->>+Gateway: POST /v1/score (Payload)
    Gateway->>+Sentinel: (Forward Request)

    Sentinel->>+Cognito: POST /analyze/identity
    Sentinel->>+Praxis: POST /analyze/behavior
    Sentinel->>+Nexus: POST /query/links

    Cognito-->>-Sentinel: Response (Score: 20)
    Praxis-->>-Sentinel: Response (Score: 75)
    Nexus-->>-Sentinel: Response (Score: 90)

    Sentinel->>Sentinel: Aggregate scores (e.g., weighted average)
    Sentinel-->>-Gateway: Final Response (Score: 85, Decision: REVIEW)
    Gateway-->>-Merchant: (Forward Final Response)
```

5. Deployment & Scalability Plan

- **Containerization:** All services will be containerized using Docker.
- **Orchestration:** Deployed on a Kubernetes (K8s) cluster (e.g., EKS, GKE, AKS).

- **Scalability:** Each microservice will have its own K8s deployment with Horizontal Pod Autoscalers (HPAs) configured.
    - `sentinel-core` will scale based on CPU and memory usage.
    - `cognito-analyzer` (which might use GPUs for ML models) can be deployed on a separate node pool with GPU instances.
- **Databases:** Use managed database services for scalability and reliability (e.g., Amazon Neptune for Graph, Redis Labs for Redis, RDS/Cloud SQL for PostgreSQL).
- **Monitoring:** Implement robust monitoring using Prometheus for metrics, Grafana for dashboards, and Jaeger/OpenTelemetry for distributed tracing to diagnose latency issues across services.

This LLD provides a concrete, multi-layered plan to build Project Chimera. It breaks down the high-level "agents" into implementable microservices with clear responsibilities, technologies, and communication patterns, ensuring the final system is not only powerful but also scalable, resilient, and maintainable.