**FraudShield: Complete Non-ML Fraud Detection Strategy (Documentation)**

**Author:** Kakashi  
**Purpose:** This document outlines the full design and strategy for FraudShield's fraud detection system without machine learning, including the "why", "how", and real-world examples to ensure clear and persistent memory of this project. It includes algorithms, rule logic, and technical integrations.

**1. Objective**

FraudShield is a stateless, real-time fraud detection platform for online sellers. It is designed to:

* Analyze transaction/user data
* Return a fraud score, decision (fraud/suspicious/not fraud), and reason list
* Operate without machine learning (ML)
* Avoid storing sensitive customer data permanently (at MVP stage)
* Be embeddable into store owners' websites via a JS script

**Supported Detection Use Cases:**

* Price tampering
* VPN/proxy abuse
* BIN/card type fraud
* Reused/stolen card numbers
* Disposable/spoofed emails
* Location mismatch (IP vs billing)
* Bot/script abuse via fast checkout
* Device fingerprint repetition

**2. Core Detection Strategy**

We are using **rule-based detection**, **real algorithms**, and **statistical + behavioral anomaly logic**.

**Chosen Architecture:**

* **IF-ELSE + Risk Scoring**: Each rule assigns a weighted score
* **Z-Score / Anomaly Check**: For timing or unusual behaviors
* **Bloom Filters**: For memory-efficient history tracking (card reuse, device ID)
* **Graph Traversal (Later)**: For finding links between accounts/devices/IPs

This hybrid strategy ensures we:

* Stay fast and transparent
* Avoid black-box models
* Are easy to tune/debug

**3. Algorithm Components with Examples**

**3.1 Aho-Corasick Automaton**

**Used For:** Fast matching of patterns (email domains, BINs, IPs)

**Example:**  
Email: user@tempmail.com matches tempmail.com in 10,000+ pattern list in one pass.  
**Result:** Disposable email detected.

**3.2 GeoDistance (Great-Circle)**

**Used For:** Calculate distance between IP geolocation and billing address

**Example:**  
Billing country = India, IP geolocates to Russia (5,000+ km apart).  
**Result:** Mismatch flag raised.

**3.3 Levenshtein Distance**

**Used For:** Fuzzy matching for typos/fraud-like inputs

**Example:**  
Domain: paypa1.com vs paypal.com → 1-char change.  
**Result:** Spoofed brand suspected.

**3.4 Z-Score Checkout Timing**

**Used For:** Detect anomalously fast (bot-like) checkouts

**Example:**  
Average checkout = 45s. User submits in 3s → z = -3.0  
**Result:** Bot behavior likely.

**3.5 Bloom Filter for Device/Card Reuse**

**Used For:** Detect reuse of same hashed card/device ID

**Example:**  
Same device\_hash\_abc123 used in 4 previous flagged transactions.  
**Result:** Device reuse → high risk.

**3.6 Graph Traversal (optional, phase 2)**

**Used For:** Discover fraud rings across users/IPs/cards

**Example:**  
3 users use same phone, same IP, and share one card.  
**Result:** Graph cluster exposes collusion.

**4. Rule Table (With Scores & Logic)**

| **Rule** | **Logic/Trigger** | **Score** |
| --- | --- | --- |
| Disposable Email | Email domain matches list | +0.4 |
| Price Tampering | Abs(Expected - Submitted)/Expected > 0.5 | +0.5 |
| IP ≠ Billing Country | Different countries or GeoDistance > 3000km | +0.6 |
| VPN/Proxy Use | Detected by external API | +0.7 |
| Checkout Speed < 6s | Time from load to click | +0.3 |
| Invalid Phone / Empty | Format error or blank | +0.2 |
| Suspicious BIN | Matches BIN blacklist or mismatched country | +0.5 |
| Reused Device ID | Found in previous fraud attempts | +0.6 |
| Reused Card Hash | Card used in >1 IP/device combo | +0.6 |
| User-Agent Spoofing | Generic/minimal user agents (e.g. curl, python-requests) | +0.3 |
| Language vs Site Mismatch | Browser language ≠ site language | +0.2 |

**Scoring Result Interpretation:**

* ≥ 0.8 → Fraud
* 0.5 – 0.79 → Suspicious ("chance")
* < 0.5 → Not Fraud

**5. JavaScript Snippet Integration Plan**

A JavaScript snippet is injected into store owner’s checkout page. It will:

* Extract DOM fields:
  + Email, phone, price, billing country, payment method
* Track checkout time from page load
* Collect browser info, language, timezone, fingerprint
* Send all to backend via fetch()

**Store Owner Action:**

* Just paste script
* Do nothing with the result — we ONLY return fraud data

**6. Fake Checkout Page (Testing Environment)**

**Purpose:**

* Simulate checkout behavior
* Feed test transactions
* Visualize fraud detection output

**Includes:**

* Price field (shown and hidden)
* Form fields: email, phone, billing, etc.
* JS script included to trigger fraud API
* Displays returned JSON:

{

"fraud\_score": 0.87,

"is\_fraud": true,

"reasons": [

"Disposable email domain",

"Price tampering detected",

"IP and billing country mismatch",

"VPN usage",

"Checkout too fast"

]

}

**7. Final Notes / Future Plan**

* ✅ **No machine learning for now** — logic is transparent and testable.
* ✅ **We only check and return**, no block/ban/notify
* ✅ **No permanent user data storage yet**, just logs
* ✅ **Real algorithms**, not gimmicks

**Later:**

* Add Redis/Mongo for fingerprint/card tracking
* Expand graph correlation
* Introduce optional ML models after result storage

**Summary**

FraudShield is designed for **practical, real-time fraud detection** using deterministic and explainable methods. This document contains the full rationale, architecture, algorithm choices, detection logic, examples, and rollout strategy. It is meant to serve as the permanent reference for what this project is, how it works, and why it was built this way.

**KEEP THIS DOCUMENT. DO NOT DELETE.**