Anti-Money Laundering Logic Code Quality Analysis

Scope

This document presents a code quality analysis sample for key logic in an Anti-Money Laundering (AML) system. It highlights strengths, weaknesses, and suggests improvement areas based on code review best practices.

1. Sample AML Detection Logic

Python Example:

```python
def is_suspicious_transaction(transaction):
    if transaction.amount > 10000 and transaction.country != "US":
        return True
    if transaction.origin_account in blacklisted_accounts:
        return True
    if transaction.frequency > 5 and transaction.period_days < 2:
        return True
    return False
```

2. Code Quality Analysis

Strengths

- Logic is straightforward and readable.
- Conditions are clearly separated.
- Early returns simplify control flow.

Weaknesses

- Hardcoded thresholds reduce flexibility.
- Lack of input validation and type checks.
- Magic values (e.g., `10000`, `5`, `2`).
- No logging or traceability on rule match.

Improvement Areas

- Extract thresholds and blacklists to configurable parameters.
- Add input validation and error handling.
- Add inline documentation and logging for auditability.
- Replace magic numbers with named constants.

3. Enhanced Example

```
THRESHOLD_AMOUNT = 10000
COUNTRY_DOMESTIC = "US"
MAX_FREQUENCY = 5
MIN_PERIOD_DAYS = 2
def is_suspicious_transaction(transaction, blacklisted_accounts):
    if not hasattr(transaction, "amount") or not hasattr(transaction, "origin_account"):
        # Handle incomplete data
        return False
    if transaction.amount > THRESHOLD_AMOUNT and transaction.country != COUNTRY_DOMESTIC:
        # Large, international transfer
        return True
    if transaction.origin_account in blacklisted_accounts:
        # Transaction originates from known blacklisted account
        return True
    if transaction.frequency > MAX_FREQUENCY and transaction.period_days < MIN_PERIOD_DAYS:
        # High-frequency, short-period activity
        return True
    return False
```

4. Recommendations

- Adopt configuration-driven rule thresholds for maintainability.
- Improve test coverageâ€”use unit tests for all logic paths.
- Integrate detailed logging and alerting on suspicious detections.
- Review and refactor code for security, clarity, and extensibility.