Security Design

Bank Vault: High-Concurrency Client-Server Network Service System

This section documents the security rationale and mechanisms integrated into the Bank Vault service. The service uses a custom binary application-layer protocol (no HTTP/WebSocket) and is built around a multi-process server with IPC-backed shared state. Security features are designed to be demonstrable, testable, and suitable for a systems programming course.

1. Threat Model

- Passive sniffing: an attacker can observe traffic and attempt to read credentials or transaction data.
- Active tampering: an attacker can inject, modify, replay, or reorder packets.
- Abuse and resource exhaustion: attackers may open many connections or send malformed packets to crash workers or degrade service.
- Unauthorized operations: attackers may guess account IDs or attempt to perform actions without proper permission.

2. Protocol-Level Security Overview

Security controls are applied directly at the application layer. The protocol supports per-packet integrity checks (CRC32 and optional MAC), optional payload encryption, authentication handshake, and replay protection via sequence numbers.

2.1 Packet framing (reference)

```
Header fields (example): Len(4) | Magic(2) | Ver(1) | Flags(1) | Op(2) | Seq(4) | Timestamp_ms(8) | CRC32/MAC | Body.
```

Flags may include: encrypted body, error response, or reserved bits for future extensions.

3. Integrity and Anti-Tamper

3.1 CRC32 (baseline)

Each packet includes CRC32 over Header (CRC field set to 0 during calculation) plus Body. CRC32 is lightweight and detects accidental corruption during transmission, and it is easy to demonstrate during testing.

3.2 Keyed MAC (recommended upgrade)

To provide real integrity against active attackers, add a keyed Message Authentication Code (MAC) per packet using a session key derived after login. Options: HMAC-SHA256 (standard) or SipHash-2-4 (fast keyed hash). With a MAC enabled, packets modified in transit are rejected without revealing plaintext.

4. Confidentiality (Encryption)

4.1 XOR encryption (minimum)

A simple XOR stream cipher can be applied to the Body using a per-session key. This prevents plaintext exposure to casual sniffing. Because XOR is not cryptographically strong, it is treated as a minimum requirement or demo mode.

4.2 Strong encryption mode (bonus)

Add an optional secure mode using AES-CTR or ChaCha20 for encrypting the Body. Include a per-packet nonce/IV (never reused for the same key). Nonce can be constructed from a random session salt + Seq.

5. Authentication

5.1 Login handshake

A simple login handshake establishes an authenticated session and a session key used for MAC/encryption. Only PING/LOGIN messages are allowed before authentication.

5.2 Challenge-response (prevents password replay)

Avoid sending the plaintext password. Suggested flow: Client sends LOGIN_HELLO(user). Server responds with CHALLENGE(server_nonce). Client replies LOGIN_PROOF = HMAC(password_hash, server_nonce || client_nonce || user). Server verifies and returns LOGIN_OK(session_id).

5.3 Password storage (server)

Store salted hashes (salt + SHA256(salt || password)) rather than plaintext. If allowed, use a slow KDF (bcrypt/argon2) for better resistance to offline cracking.

6. Replay and Reordering Protection

Each authenticated connection/session maintains a monotonically increasing sequence number. The server rejects duplicates and out-of-window sequence numbers. Optionally enforce timestamp windows (e.g., accept only if within 30 seconds of server time) to reduce replay feasibility.

7. Authorization (Access Control)

After authentication, the server binds requests to the authenticated user/session. Account operations must pass ownership checks. For example, WITHDRAW/DEPOSIT/BALANCE require the account to be owned by the logged-in user; TRANSFER requires the source account to be owned.

8. Robust Parsing and Input Validation

- Enforce MAX_PACKET; reject lengths smaller than header size or larger than configured limit.
- Validate Magic and Version before parsing Body.
- Validate opcode, and restrict opcodes permitted before login.

8. Robust Parsing and Input Validation (continued)

- Validate all variable-length fields against remaining bytes (no over-read).
- Use safe memory APIs; avoid unbounded string functions; check integer overflow for amounts and balances.
- Treat repeated invalid packets as suspicious and disconnect after a threshold.

9. Abuse Resistance (DoS and Brute Force)

- Per-connection and per-IP rate limiting (token bucket).
- Login throttling: exponential backoff or temporary lockouts after repeated failures.
- Connection caps: max connections total and max per IP; reject gracefully with SERVER_BUSY.
- Worker isolation: multiprocess design limits blast radius if a worker crashes; master can respawn workers.

10. Audit Logging and Forensics

Log security-relevant events in structured form: login success/failure, transactions, invalid MAC/checksum, replay detections, and rate limiting triggers. Do not log plaintext passwords. For bonus tamper-evidence, maintain a rolling hash chain over log entries (hash[i] = SHA256(hash[i-1] || entry[i])).

Appendix A. Status and Error Codes

Status codes are returned in responses as a uint16 field. 0x0000 indicates success; non-zero values indicate errors. The client uses these codes to decide whether to retry, back off, re-authenticate, or close the connection.

Range	Category	Examples
0x0000	Success	OK
0x1xxx	Protocol / Parse	BAD_MAGIC, BAD_LENGTH, BAD_MAC, REPLAY
0x2xxx	Auth / Session	NOT_AUTH, AUTH_FAILED, SESSION_INVALID, PERMISSION
0x3xxx	Business Logic	NO_SUCH_ACCOUNT, INSUFFICIENT_FUNDS, AMOUNT_INVALID
0x4xxx	Rate / Availability	RATE_LIMIT, TOO_MANY_CONN, SERVER_BUSY, TIMEOUT
0x5xxx	Internal Server	INTERNAL, IPC_FAIL, SHUTTING_DOWN

Detailed Codes

Code	Name	Meaning / Suggested action
0x0000	OK	Request succeeded.
0x1001	ERR_BAD_MAGIC	Magic mismatch; not our protocol. Close connection.
0x1002	ERR_BAD_VERSION	Unsupported protocol version. Close connection.
0x1003	ERR_BAD_LENGTH	Packet length invalid/inconsistent. Close connection.
0x1004	ERR_TOO_LARGE	Packet exceeds MAX_PACKET. Close connection.
0x1005	ERR_BAD_OPCODE	Unknown/unsupported opcode.
0x1006	ERR_PARSE	Malformed body or field mismatch.
0x1007	ERR_BAD_CHECKSUM	CRC32 invalid.
0x1008	ERR_BAD_MAC	MAC verification failed. Close connection recommended.
0x100A	ERR_REPLAY	Duplicate/old sequence detected.
0x100C	ERR_BAD_TIMESTAMP	Timestamp outside allowed window.
0x2001	ERR_NOT_AUTH	Operation requires login.
0x2002	ERR_AUTH_FAILED	Invalid credentials/proof; count failure.
0x2003	ERR_AUTH_LOCKED	Temporarily locked due to brute-force protection.
0x2004	ERR_SESSION_INVALID	Session expired/unknown; re-login.
0x2006	ERR_PERMISSION	Access denied: account ownership or role check failed.

0x3001	ERR_NO_SUCH_ACCOUNT	Account ID does not exist.
0x3002	ERR_INSUFFICIENT_FUNDS	Insufficient funds for withdraw/transfer.
0x3003	ERR_AMOUNT_INVALID	Amount <= 0 or violates rules.
0x3006	ERR_TXN_CONFLICT	Lock/contention prevented operation; retry with backoff.
0x4001	ERR_RATE_LIMIT	Too many requests; backoff and retry.
0x4002	ERR_TOO_MANY_CONN	Server connection cap reached; retry later.
0x4003	ERR_SERVER_BUSY	Server busy/queue full; retry later.
0x4004	ERR_TIMEOUT	Request timed out.
0x5004	ERR_SHUTTING_DOWN	Server shutting down gracefully.

Appendix B. Client Handling Guidance

Recommended client behavior improves stability during stress testing and makes failure modes explicit in demos.

- Close connection immediately on: ERR_BAD_MAGIC, ERR_BAD_LENGTH, ERR_TOO_LARGE, ERR_BAD_MAC, ERR_DECRYPT.
- Re-authenticate on: ERR_NOT_AUTH, ERR_SESSION_INVALID.
- Retry with exponential backoff on: ERR_SERVER_BUSY, ERR_TIMEOUT, ERR_TXN_CONFLICT, ERR_RATE_LIMIT.
- If ERR_AUTH_LOCKED is returned, honor retry_after_ms (if provided) and avoid rapid reconnect loops.

Suggested secure defaults

- MAX_PACKET = 65536 bytes; reject larger packets.
- Enable MAC for authenticated sessions; keep CRC32 for debug only.
- Require login for all state-changing operations.
- Rate limit: 50 req/s per connection (tunable); enforce connection caps per IP.
- Disconnect after N malformed packets (e.g., N=3) to prevent parser abuse.