Security Test: Vulnerability Analysis Report

Security Assessment Submission

November 2025

Executive Summary

This report presents a comprehensive security analysis identifying **11 critical to medium vulnerabilities** across secrets management, injection attacks, encryption, input validation, and logging practices.

All identified vulnerabilities have been remediated in the accompanying fixed code.

Key Findings: - 5 Critical vulnerabilities - 4 High vulnerabilities - 2 Medium vulnerabilities - All issues mapped to CWE/OWASP standards - Complete code remediation provided

---

1. Vulnerability Summary Table

ID	Issue	Severity	CWE/OWASP	Location
V-01	Hardcoded Secrets	Critical	CWE-798	Lines 14-18
V-02	Secrets in Logs	Critical	CWE-532	Lines 31-32, 62, 131, 160
V-03	Disabled SSL/TLS	Critical	CWE-295	Lines 36, 40, 96, 177
V-04	SQL Injection	Critical	CWE-89	Lines 71, 172
V-05	Plaintext Data	Critical	CWE-256	Lines 49-58
V-06	No Input Validation	High	CWE-20	Lines 65-71, 163-183
V-07	Insecure HTTP	High	CWE-319	Lines 25, 177
V-08	No Rate Limiting	High	CWE-770	Lines 83-107
V-09	Broad Exceptions	Medium	CWE-396	Multiple
V-10	No DB Access Control	High	CWE-732	Lines 49-58
V-11	Hardcoded Config	Medium	CWE-547	Multiple

---

2. Detailed Vulnerability Analysis

V-01: Hardcoded Secrets in Source Code

Description: Multiple credentials hardcoded as plaintext constants (lines 14-18)

Impact: CRITICAL - Complete compromise if code committed to version control

Evidence:

```
API_KEY = "sk-1234567890abcdef1234567890abcdef"
DATABASE_PASSWORD = "admin123"
AWS_ACCESS_KEY = "AKIAIOSFODNN7EXAMPLE"
```

Fix: Use environment variables

```
API_KEY = os.getenv('API_KEY')
DATABASE_PASSWORD = os.getenv('DATABASE_PASSWORD')
```

Verification: Search codebase for hardcoded patterns, use secret scanning tools

---

V-02: Secrets Exposed in Application Logs

Description: Credentials logged at DEBUG and ERROR levels

Impact: CRITICAL - Secrets accessible to anyone with log access

Evidence:

```
self.logger.info(f"Initializing with API key: {API_KEY}")
self.logger.error(f"S3 upload failed | Credentials: {AWS_ACCESS_KEY}")
```

Fix: Remove all credential logging, implement log filtering

Verification: Run app and grep logs for sensitive patterns

V-03: SSL/TLS Certificate Validation Disabled

Description: All HTTPS requests bypass certificate validation

Impact: CRITICAL - Vulnerable to Man-in-the-Middle attacks

Evidence:

```
self.session.verify = False
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
```

Fix: Enable validation

```
self.session.verify = True  # Default, explicitly set
```

Verification: Use mitmproxy to test - connection should fail with invalid cert

---

V-04: SQL Injection Vulnerability

Description: Unsanitized input concatenated into SQL queries

Impact: CRITICAL - Data exfiltration, authentication bypass, DoS

Evidence:

```
query = f"SELECT * FROM user_data WHERE id = {user_id}"
```

Fix: Use parameterized queries

```
query = "SELECT * FROM user_data WHERE id = ?"
cursor.execute(query, (user_id,))
```

Verification: Test with malicious inputs like `"1 OR 1=1"`

V-05: Sensitive Data Stored in Plaintext

Description: Passwords, credit cards, SSN stored unencrypted

Impact: CRITICAL - PCI-DSS/GDPR violations, identity theft

Evidence:

```
CREATE TABLE user_data (
    password TEXT,      -- Plaintext
    credit_card TEXT,   -- Plaintext
    ssn TEXT           -- Plaintext
)
```

Fix: Hash passwords with bcrypt, encrypt sensitive fields

```
password_hash = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
```

Verification: Query database - passwords should be hashes, not plaintext

---

V-06: Missing Input Validation

Description: No validation on user inputs

Impact: HIGH - Enables SQL injection, type confusion, DoS

Evidence:

```
def fetch_user_data(self, user_id):
    query = f"SELECT * FROM user_data WHERE id = {user_id}"  # Direct use
```

Fix: Validate all inputs

```
if not isinstance(user_id, int) or user_id < 1:
    raise ValueError("Invalid user_id")
```

Verification: Fuzz with malformed inputs, ensure proper errors

---

V-07: Insecure HTTP Communication

Description: Webhook endpoint uses HTTP instead of HTTPS

Impact: HIGH - Data transmitted in cleartext

Evidence:

```
WEBHOOK_ENDPOINT = "http://internal-webhook.company.com/process"
```

Fix: Use HTTPS

```
WEBHOOK_ENDPOINT = "https://internal-webhook.company.com/process"
```

Verification: Capture traffic with Wireshark - should be encrypted

---

V-08: Missing Rate Limiting

Description: No rate limiting on API calls

Impact: HIGH - DoS attacks, cost explosion

Evidence:

```
def call_external_api(self, data):
    # No rate limiting
```

Fix: Implement token bucket rate limiter

Verification: Load test with 1000 requests - should rate limit

---

V-09: Overly Broad Exception Handling

Description: Generic Exception catching hides security issues

Impact: MEDIUM - Masked bugs, poor observability

Evidence:

```
except Exception as e:  # Too broad
```

Fix: Catch specific exceptions

```
except sqlite3.Error as e:
    # Specific handling
```

Verification: Trigger various exception types, verify proper handling

---

V-10: Missing Database Access Controls

Description: No row-level security, permissions, or encryption

Impact: HIGH - Privilege escalation, data leakage

Fix: Implement database roles, row-level security, column encryption

Verification: Test with restricted user - should not access all data

---

V-11: Hardcoded Configuration Values

Description: URLs, regions, servers hardcoded in source

Impact: MEDIUM - Can’t run in multiple environments

Evidence:

```
DB_CONNECTION_STRING = "postgresql://admin:...@prod-db.company.com:5432"
region_name='us-east-1'  # Hardcoded
```

Fix: Use environment variables

```
DB_HOST = os.getenv('DB_HOST', 'localhost')
AWS_REGION = os.getenv('AWS_REGION', 'us-east-1')
```

Verification: Deploy to dev, staging, prod with different configs

---

3. Summary of Remediations

Code Changes Overview

The fixed code implements these security improvements:

Secrets Management

✅ Environment variables replace hardcoded secrets ✅ AWS Secrets Manager support ✅ No credential logging

Encryption & Communication

✅ SSL/TLS certificate validation enabled ✅ HTTPS-only endpoints ✅ bcrypt password hashing ✅ Field-level encryption for sensitive data

Injection Prevention

✅ Parameterized SQL queries ✅ Comprehensive input validation ✅ Type checking and whitelisting

Resource Controls

✅ Token bucket rate limiting ✅ Retry logic with exponential backoff ✅ Request timeouts

Error Handling

✅ Specific exception types ✅ No sensitive data in error messages ✅ Structured logging with redaction

---

4. Deployment Checklist

Before production deployment:

- [ ] Rotate all exposed credentials

- [ ] Configure AWS Secrets Manager
- [ ] Set up environment-specific configs
- [ ] Enable database encryption at rest
- [ ] Configure rate limiting thresholds
- [ ] Set up centralized logging with field redaction
- [ ] Enable AWS GuardDuty
- [ ] Conduct penetration testing
- [ ] Configure monitoring and alerting
- [ ] Train team on secure coding practices

Conclusion

This assessment identified **11 critical to medium vulnerabilities**, all remediated following OWASP and CWE best practices. The fixed code provides a solid foundation for secure application development.

Recommendation: Deploy fixed code to staging, conduct thorough security validation before production deployment.