Security Assessment Report

Project: BrokenWebAPI (.NET 6.0) **Assessment Date:** April 17, 2025

---

1. Executive Summary

The BrokenWebAPI application was evaluated against common security vulnerabilities using static (SAST), dynamic (DAST), software composition (SCA), and container security analysis tools.

- **Overall Risk Level:** Medium
- **Critical Risks Identified:** None
- **Primary Concern Areas:** Insecure coding patterns, dependency vulnerabilities, missing security headers, and outdated framework usage.

---

2. Key Findings Overview

Category	Tool	Severity	Description
Code Vulnerabilities (SAST)	Semgrep, Snyk Code	High/ Medium	SQL Injection, Hardcoded Secrets, Insecure Deserialization
Dependency Vulnerabilities (SCA)	Snyk, Trivy	Medium/ Low	Vulnerable packages (e.g., Microsoft.IdentityModel.JsonWebTokens, APT)
Runtime / Container Issues	Trivy, Snyk Image	Low	Outdated OS packages, No .NET SDK vulnerabilities flagged automatically
Dynamic Web Security (DAST)	Netsparker, ZAP	Medium/ Low	Missing TLS, CORS Misconfiguration, Missing Security Headers (CSP, Referrer Policy)
Framework Support Issues (Manual Finding)	Manual Analysis	High	.NET 6.0 Outdated and EOL (End of Life)

---

3. Detailed Findings

3.1 Code Vulnerabilities (Semgrep & Snyk Code)

- **SQL Injection (High):** Unvalidated SQL query building found in `DatabaseService.cs`.
- **Hardcoded Secrets (High):** Sensitive data hardcoded in application configuration.
- **Deserialization of Untrusted Data (High):** BinaryFormatter and unsafe deserialization practices detected.
- **Stack Trace Exposure (Medium):** Stack traces displayed in non-production environments.

3.2 Dependency Vulnerabilities (Snyk Test + Image Scan)

- **Resource Exhaustion Vulnerability (Medium):** Outdated `Microsoft.IdentityModel.JsonWebTokens` library.
- **Multiple Low Severity OS Package Issues:** Debian 11 packages flagged with minor vulnerabilities.

3.3 Dynamic Application Testing (ZAP, Netsparker)

- **SSL/TLS Not Implemented (Medium):** No HTTPS enforced.
- **CORS Misconfiguration (Medium):** Access-Control-Allow-Origin misconfiguration.
- **Missing Security Headers (Low):** No CSP, X-Content-Type-Options, Referrer Policy headers.
- **Internal Server Error Responses (Low):** Server responds with generic 500 errors.

3.4 Framework Support Issues (Manual Finding)

- **Use of Outdated Framework:**
  - The application uses **.NET 6.0**, which reached **End of Life (EOL)** on **November 12, 2024**.
  - Continued use of unsupported frameworks exposes applications to unpatched vulnerabilities and compliance risks.
  - **Recommendation:** Migrate to **.NET 8.0 LTS** as soon as possible.
  - **Reference:** Microsoft .NET 6 Support Policy

---

4. Recommendations

Risk	Recommendation
SQL Injection	Use parameterized queries with `SqlCommand` and `SqlParameter`.
Hardcoded Secrets	Move secrets to environment variables or secret managers.
Insecure Deserialization	Avoid `BinaryFormatter`; use `System.Text.Json` or other safe serializers.
SSL/TLS	Configure HTTPS using a reverse proxy like NGINX and Let's Encrypt certificates.
CORS Policy	Restrict `Access-Control-Allow-Origin` to specific trusted domains.
Security Headers	Add missing headers like Content-Security-Policy, Referrer-Policy, X-Content-Type-Options.
Dependency Upgrades	Update `Microsoft.IdentityModel.JsonWebTokens` to version >=7.1.2.
Container Security	Monitor and update base images regularly.
Framework Upgrade	Migrate from .NET 6.0 to .NET 8.0 LTS as per Microsoft policy.

5. Conclusion

While no critical vulnerabilities were identified, several high and medium severity issues were found. Immediate remediation is advised, particularly around SQL Injection risks, hardcoded credentials, and enabling HTTPS.

Implementing fixes and integrating continuous security scanning into the CI/CD pipeline will greatly strengthen the application's security posture.