코드 취약점 분석 보고서

2026. 03. 25

3조_E둘I들
팀원: 김민수, 배민기, 이수진, 최하경

1. 코드 보안 취약점 진단

가. 개요

본 보고서는 KISIA_API 프로젝트의 소스 코드 및 실습 가이드를 분석하여, 시스템에 존재하는 주요 보안 취약점을 식별하고 그 원인을 분석한다. 또한 이를 기반으로 실제 서비스 환경에서 적용 가능한 보안 개선 방안을 제시하는 것을 목적으로 한다. 해당 프로젝트는 취약한 API 서버(vulnerable-server)와 보안이 강화된 서버(secure-server)를 비교하는 구조로 구성되어 있으며, 다양한 API 보안 취약점 사례를 실습을 통해 확인할 수 있다.

나. 보안 진단 방법과 진단 도구

본 진단은 소스 코드의 취약점을 식별하기 위해 다음과 같은 단계별 방법론을 적용하였다.

① 정적 분석 (SAST - Static Application Security Testing)

코드를 실행하지 않은 상태에서 소스 코드 자체를 분석하여 잠재적인 보안 약점을 찾아내는 방식으로, 입력값 검증 부재 (SQL Injection, XSS 패턴 검색)를 통한 취약점인 위험한 한수 사용, 하드코딩된 비밀번호나 API 키 존재 여부 등을 검사하였다.

② 동적 분석 (DAST - Dynamic Application Security Testing)

서버를 실제로 구동시킨 후, 웹 요청을 변조하여 서버의 응답을 확인하는 블랙박스 방식을 채택하였다. 특수문자 입력 시의 에러 발생 여부와 권한이 없는 사용자의 페이지 접근 가능 여부(IDOR), 세션 관리 및 인증 우회 테스트를 통해 검사하였다.

다. 보안 진단 결과 총평

① BOLA (Broken Object Level Authorization)

해당 취약점은 사용자 ID 또는 리소스 ID를 변경하는 것만으로 타인의 데이터에 접근 가능하다. 해당 취약점의 근거로는 /api/orders/{id} 요청 시에 Alice의 토큰으로 Bob의 주문 정보에 조회 가능하다는 문제점이 존재한다.

해당 취약점에 대한 원인으로는 요청한 사용자와 해당 리소스의 소유가 간의 관계를 검증하는 로직이 부재하다는 것이었다. 해당 취약점으로 인해 발생할 수 있는 위험으로는 공격자가 다른 사용자의 주문, 개인정보 등 민감 데이터를 무단으로 열람 가능하다는 위험이 존재했다.

② 부적절한 인가 처리 (Function Level Authorization)

부적절한 인가 처리는 일반 사용자 계정으로 관리자 API 접근 및 사용자 정보 수정과 삭제가 가능한 취약점이다. 해당 취약점의 소스 코드 내 근거로는 /api/admin/users, /api/users/{id} 등에 일반 사용자로 접근 가능하다. 이는 역할(Role) 기반 접근제어 (RBAC)의 미구현 또는 검증 로직의 누락이 원인으로 작용하며, 해당 취약점을 통해 권한 상승(Privilege Escalation)을 통해 시스템 전체 제어가 가능한 위험이 작용할 수 있다.

③ 과도한 데이터 노출 (Excessive Data Exposure)
사용자 조회 API에서 password 또는 내부의 메타 데이터가 노출될 수 있는 취약점으로, 이는 /api/users/me 응답에 password_hash 및 생성/수정 정보를 포함한다는 사실에서 기인한다. 데이터 조회 시 필요한 필드만 선택하지 않고 전체 데이터를 반환(SELECT * 사용)하여 민감 정보 유출로 인한 계정 탈취 및 시스템 구조가 노출될 위험이 있다.

④ Mass Assignment 취약점
해당 취약점은 클라이언트에서 전달한 요청 데이터가 그대로 데이터베이스에 반영된다. 사용자 정보 수정시에 'role: 'admin'값 삽입으로 권한 상승이 가능한 코드가 존재하였다. 이는 입력 데이터에 대한 화이트리스트 검증값의 부재로 인해 발생하는 것이며, 일반 사용자가 관리자 권한을 획득하여 시스템을 장악할 수 있다는 위험 요소가 존재한다.

⑤ 입력 값 검증 미흡 (Lack of Input Validation)
이는 잘못된 데이터가 서버에서 정상 처리되어 발생하는 취약점으로, /api/orders에서 qauntity : -1과 같은 비정상적인 값을 허용하는 코드에서 취약점이 발생한다. 이는 입력 데이터에 대한 타입, 범위, 형식 검증 로직의 미구현으로 인해 기인하는 취약점이므로 비즈니스 로직 오류와 데이터 무결성 훼손, 추가적인 공격이 가능하다.

⑥ SQL Injection (SQL 인젝션)
사용자 입력값이 데이터베이스 쿼리 구조를 조작해 인증 우회하거나 데이터를 탈취한다. 해당 취약점은 routes/auth.js 내의 로그인 로직에서 입력받은 email, password 검증 없이 SQL문자열에 직접 결합하여 사용하는 로직에서 찾을 수 있었다.
이는 사용자 입력 데이터에 대한 필터링 부재 및 Prepared Statement의 미사용으로 인해 발생하는 것으로, 공격자가 admin'- 과 같은 구문을 입력하여 비밀번호 검증 없이 관리자 계정으로 로그인 가능하며, DB 내의 전체 데이터 유출 및 삭제의 위험이 있다.

⑦ Reflected Cross-Site Scripting (XSS)
XSS 취약점은 API 응답 메시지에 포함된 악성 스크립트가 사용자의 브라우저에서 실행되는 것으로, 검색 결과나 에러 메시지 변환 시에 사용자가 입력한 파라미터 값을 그대로 HTML 응답에 포함하여 출력하는 곳에서 기인했다. 해당 취약점의 원인으로는 출력값에 대한 적절한 HTML Entity 인코딩 처리의 미비함이었고, 이를 통해 사용자의 세션 쿠키 탈취(Session HiJacking), 피싱 사이트로의 리다이렉트, 또는 악성 스크립트 실행을 통한 사용자 기기 제어가 가능해진다.

취약점을 표로 정리할 경우 [표 1-1] 과 같다.

3. 개선 방안

① 소유권 검증 로직 도입 (BOLA 대응)
요청 사용자와 리소스 소유자의 ID를 비교하는 로직을 구현하고, 미들웨어를 통해 공통

취약점	현상	원인	근거	위험
BOLA	사용자 ID를 변경하는 것만으로 타인의 데이터에 접근 가능	/api/orders/{id} 요청 시, Alice의 토큰으로 Bob의 주문 정보 조회가능	요청한 사용자와 해당 리소스의 소유자 간의 관계를 검증하는 로직 부재	공격자가 다른 사용자의 주문, 개인정보 등 민감 데이터를 무단으로 열람 가능
부적절한 인가 처리	일반 사용자 계정으로 관리자 API 접근 및 사용자 정보 수정/삭제 가능	/api/admin/users, /api/users/{id} 등에 일반 사용자로 접근 가능	역할 기반 접근 제어(RBAC) 미구현 또는 검증 로직 누락	권한 상승을 통해 시스템 전체 제어 가능
과도한 데이터 노출	사용자 조회 API에서 password 또는 내부 메타데이터 노출 가능	/api/users/me 응답에 password_hash 및 생성/수정 정보 포함 가능	데이터 조회 시 필요한 필드만 선택하지 않고 전체 데이터를 반환 (SELECT * 사용)	민감 정보 유출로 인한 계정 탈취 및 시스템 구조 노출
Mass Assignment 취약점	클라이언트에서 전달한 요청 데이터가 그대로 데이터베이스에 반영됨	사용자 정보 수정 시 role: 'admin' 값 삽입으로 권한 상승 가능	입력 데이터에 대한 화이트리스트 검증 부재	일반 사용자가 관리자 권한을 획득하여 시스템을 장악할 수 있음
입력 값 검증 미흡	잘못된 데이터가 서버에서 정상 처리됨	/api/orders 요청 시 quantity: -1과 같은 비정상 값 허용	입력 데이터에 대한 타입, 범위, 형식 검증 로직 미구현	비즈니스 로직 오류, 데이터 무결성 훼손, 추가적인 공격 가능
SQL Injection	사용자 입력값이 데이터베이스 쿼리 구조를 조작, 인증 우회 / 데이터 탈취	routes/auth.js 내의 로그인 로직에서 입력받은 값 검증 없이 SQL 문자열에 직접 결합하여 사용	사용자 입력 데이터에 대한 필터링 부재 및 Prepared Statement 미사용.	공격자가 admin' - 과 같은 구문을 입력하여 비밀번호 검증 없이 관리자 계정으로 로그인 가능하며, DB 내의 전체 데이터 유출 및 삭제 위험 있음.

XSS	API 응답 메시지에 포함된 악성 스크립트가 사용자의 브라우저에서 실행	검색 결과나 에러 메시지 반환 시, 사용자가 입력한 파라미터 값을 그대로 HTML 응답에 포함하여 출력	출력 값에 대한 적절한 HTML Entity 인코딩 처리 미비	사용자의 세션 쿠키 탈취, 피싱 사이트로의 리다이렉트, 또는 악성 스크립트 실행을 통한 사용자 기기 제어 가능

[표 1-1] 보안 진단 결과 총평

검증을 처리한다. 권한 없는 접근 시에 404 혹은 403 응답을 반환하는 로직을 도입하여 BOLA 취약점에 대응할 수 있도록 한다.

② 역할 기반 접근 제어(RBAC) 구현
authorize('admin')와 같은 미들웨어를 통해 관리자 API를 보호하고, 사용자 역할(Role)에 따른 접근 정책의 동의 로직을 추가하여 사용자 역할에 따라 권한 및 인증, 인가를 제한하도록 한다. 해당 로직을 구현할 때는 최소 권한 원칙(Principle of Least Privilege)를 적용하여 구현할 수 있도록 한다.

③ 응답 데이터 필터링
패스워드와 내부 메타데이터를 제외하고 필요한 데이터만 선택적으로 반환할 수 있도록 응답 데이터를 필터링하는 로직을 추가한다. 해당 로직을 구현할 때는 DTO또는 Serializer 패턴을 적용하고 민감 정보는 API 응답에서 완전히 제거할 수 있도록 한다,

④ 입력 값 검증의 강화
Joi와 같은 유효성 검사 라이브러리를 활용하여 입력 데이터를 검증할 수 있도록 한다. 데이터 타입, 길이, 범위, 패턴을 명확히 정리하며 잘못된 요청은 400 Bad Request로 처리할 수 있도록 구현함을 원칙으로 한다.

⑤ Mass Assignment 방어
허용된 필드만 처리하는 화이트리스트 기반 로직을 구현하여 명시적으로 처리할 수 있도록 한다. 민감 필드는 서버 내부에서만 관리함을 원칙으로 구현한다.

⑥ API 보안 강화 정책 적용
JWT 토큰 검증의 미들웨어를 강화하고, 강화 시엔 만료 시간과 서명 검증을 포함하여 강화한다. Rate Limiting을 적용하고 HTTPS(TLS) 적용으로 데이터 전송 구간을 암호화한다.

4. 결론

KISIA_API 프로젝트는 다양한 API 보안 취약점을 실습 목적으로 포함하고 있으며, 실제 서비스 환경에서 발생할 수 있는 대표적인 보안 문제를 잘 보여준다.
특히 BOLA, 권한 검증 부재, 데이터 노출, Mass Assignment와 같은 취약점은 실제 서비스에서도 매우 빈번하게 발생하는 문제로, 이에 대한 적절한 대응이 필수적이다.
본 보고서에서 제시한 개선 방안을 적용할 경우, 주요 취약점을 효과적으로 제거할 수 있으며, 전체 API 보안 수준을 크게 향상시킬 수 있을 것으로 판단된다.