리팩토링 44 주차

# DVWA 실습 보고서

SQL Injection, SQL Injection(blind) (Medium)

# 목차

# 1. 개요

## 1.1 프로젝트 목적

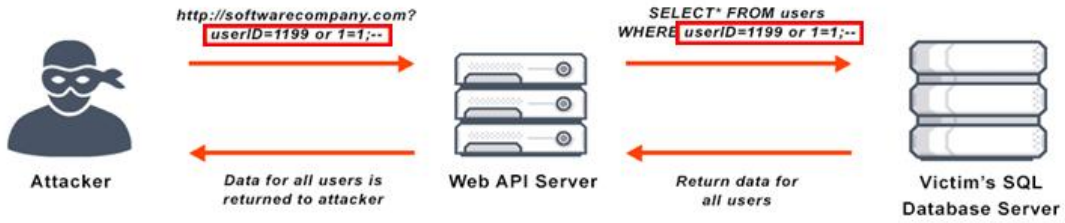
본 프로젝트는 *DVWA(Damn Vulnerable Web Application)*를 이용하여 웹 애플리케이션 취약점인 *SQL Injection*과 *Blind SQL Injection* 공격 실습과 그에 따른 공격 기법과 대응 방법을 학습하는 것을 목적으로 한다.

## 1.2 실습 환경

- VMware : 17.6.2
- Kali Linux: 2025.2
- Apache : 2.4.63
- PHP : 8.4.8
- MariaDB : 11.8.1
- DVWA : 2.5
- DVWA Security Level: Medium
- 브라우저: Firefox

# 2. SQL Injection

## 2.1 개요




[사진 1] SQL Injection 취약점

SQL Injection 취약점이란 설계된 SQL 구문에서 사용자의 입력값 검증이 미흡하여, 악의적인 명령을 실행하는 임의의 쿼리를 삽입해 공격이 가능한 취약점이다.

애플리케이션이 사용자 입력을 적절히 검증하지 않고, 입력값을 그대로 SQL 쿼리에 포함시킬 때 SQL Injection이 발생한다. 공격자는 이를 통해 데이터베이스를 조작, 열람하여 민감한 정보를 탈취할 수 있다.

## 2.2 SQL Injeciton 실습

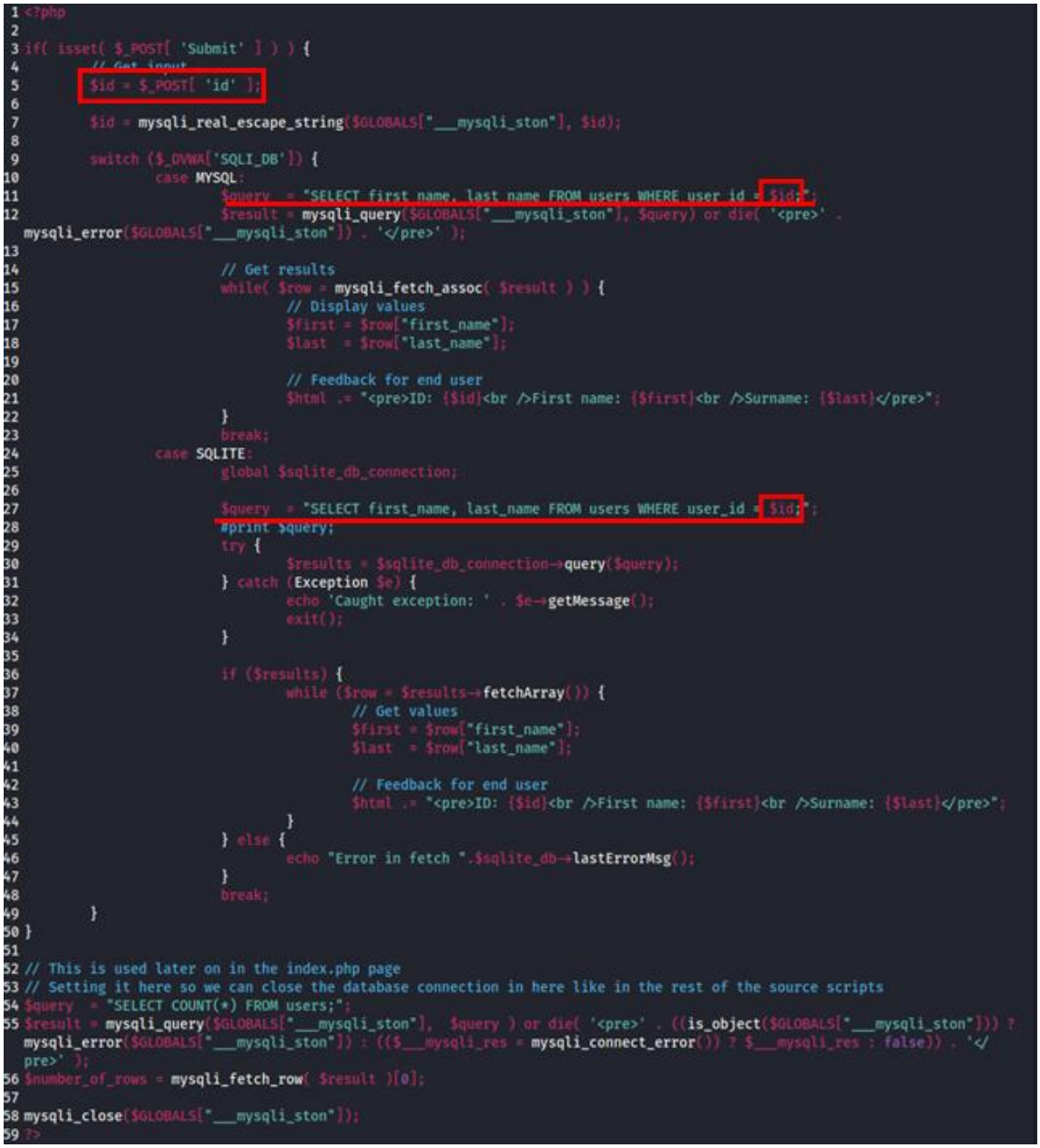

[사진 2] DVWA SQL Injection 페이지

DVWA 실습 페이지의 왼쪽 탭에서 'SQL Injection'을 눌러 SQL Injection 실습 페이지로 이동한다. 그러면 User ID를 선택할 수 있는 드롭다운 메뉴와 'Submit' 버튼이 보인다.


[사진 3] 드롭다운 메뉴와 양식 제출 결과

드롭다운 메뉴를 확인해보면 정수 1~5까지의 범위의 숫자를 선택할 수 있고 숫자를 선택하고 제출버튼을 누르면 ID에 해당하는 'First name', 'Surname' 정보를 하단에 출력해주는 것을 볼 수 있다.

Low 난이도에서는 텍스트 입력창을 통해서 입력하는 것이 가능했다. 하지만 Medium 난이도에서는 드롭다운 메뉴로 입력할 수 있는 값을 제한하고 있다.



```php
<?php

if( isset( $_POST[ 'Submit' ] ) ) {
        // Get input
        $id = $_POST[ 'id' ];

        $id = mysqli_real_escape_string($GLOBALS["___mysqli_ston"], $id);

        switch ($_DVWA['SQLI_DB']) {
                case MYSQL:
                        $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
                        $result = mysqli_query($GLOBALS["___mysqli_ston"], $query) or die( '<pre>' .
mysqli_error($GLOBALS["___mysqli_ston"]) . '</pre>' );

                        // Get results
                        while( $row = mysqli_fetch_assoc( $result ) ) {
                                // Display values
                                $first = $row["first_name"];
                                $last  = $row["last_name"];

                                // Feedback for end user
                                $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
                        }
                        break;
                case SQLITE:
                        global $sqlite_db_connection;

                        $query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
                        #print $query;
                        try {
                                $results = $sqlite_db_connection->query($query);
                        } catch (Exception $e) {
                                echo 'Caught exception: ' . $e->getMessage();
                                exit();
                        }

                        if ($results) {
                                while ($row = $results->fetchArray()) {
                                        // Get values
                                        $first = $row["first_name"];
                                        $last  = $row["last_name"];

                                        // Feedback for end user
                                        $html .= "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";
                                }
                        } else {
                                echo "Error in fetch ".$sqlite_db->lastErrorMsg();
                        }
                        break;
        }
}

// This is used later on in the index.php page
// Setting it here so we can close the database connection in here like in the rest of the source scripts
$query  = "SELECT COUNT(*) FROM users;";
$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ) or die( '<pre>' . ((is_object($GLOBALS["___mysqli_ston"])) ?
mysqli_error($GLOBALS["___mysqli_ston"]) : (($___mysqli_res = mysqli_connect_error()) ? $___mysqli_res : false)) . '</
pre>' );
$number_of_rows = mysqli_fetch_row( $result )[0];

mysqli_close($GLOBALS["___mysqli_ston"]);
?>
```

[사진 4] DVWA SQL Injection Medium 소스코드

이 페이지의 소스코드를 확인해보면 클라이언트의 'id'값을 받아서 'mysql_real_escape_string' 함수로 이스케이핑을 진행해주는 것 이외에 아무런 필터링 없이 바로 쿼리에 삽입하고 있는 것을 확인할 수 있다. 이 부분이 SQL Injection 취약점을 야기하고 있다.

Low 난이도보다 클라이언트의 입력을 자유롭게 할 수 없도록 제한했지만 서버에서 처리하는 로직은 아직 취약한 상태로 남아있다.

### 2.2.1 시나리오

#### 1) DOM 조작

서버에서는 클라이언트에서 보내준 'id' 값을 받아서 바로 SQL에 삽입하고 있다. 따라서 브라우저의 개발자 도구를 이용하여 드롭다운 메뉴의 각 옵션의 값을 수정하여 제출한다면 서버는 수정한 값을 SQL 구문에 그대로 삽입할 것이다.

1. DOM을 조작하여 드롭다운 메뉴의 값을 수정한다.
2. 수정한 값을 선택하고 제출한다.
3. 서버는 수정한 값을 그대로 SQL 구문에 삽입한다.

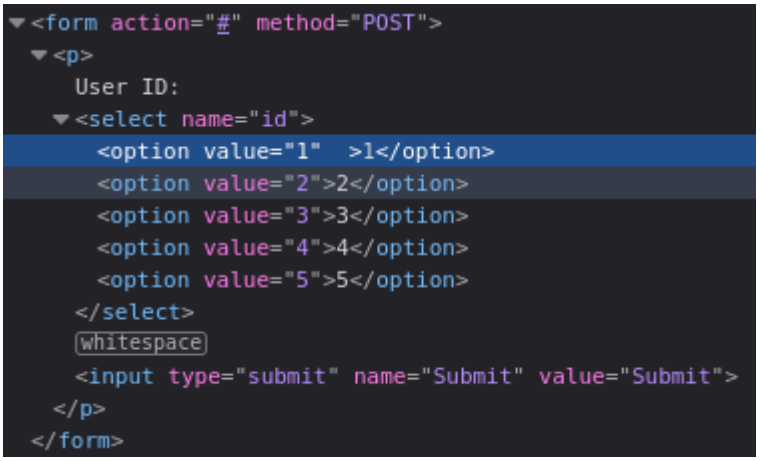
#### 2) Ruquest 조작

서버에서 사용자 입력값에 대한 검증을 하지 않기 때문에 Burp Suite와 같은 프록시 툴을 사용하여 Request의 'id'값을 수정하는 방법도 유효할 것이다.

1. 드롭다운 메뉴에서 아무 값이나 선택한 후 제출하여 서버에 요청을 보낸다.
2. Proxy를 이용하여 클라이언트의 요청을 잡은 후 'id'값을 수정한다.
3. 서버는 공격자가 수정한 값을 그대로 SQL 구문에 삽입한다

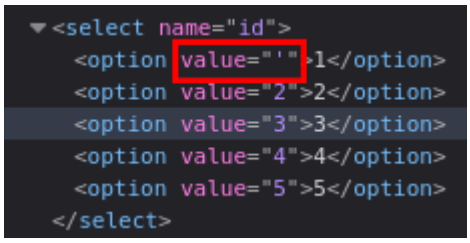
### 2.2.2 실습

#### 1) DOM 조작



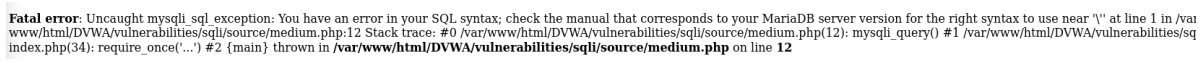

[사진 5] 페이지 개발자 도구 Inspector

브라우저의 개발자 도구를 열어 'Inspector' 메뉴에서 드롭다운 메뉴를 찾는다.




[사진 6] 드롭다운 메뉴 옵션 값 수정

먼저 SQL Injection 공격이 가능한지 판단하기 위해서 첫번째 옵션의 값을 "'"로 수정하고 제출한다.




[사진 7] 제출 오류 메시지

그러면 위와 같은 오류 메시지가 출력되는 것을 확인할 수 있다. 따라서 SQL Injection 공격이 가능하다. 오류 메시지는 아래와 같다.

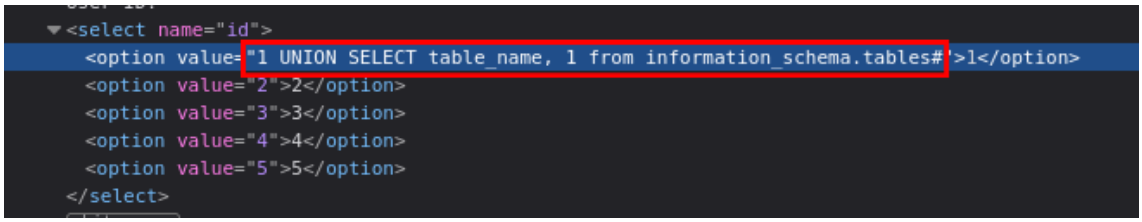
- "**Fatal error**: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '₩'' at line 1 in /var/www/html/DVWA/vulnerabilities/sqli/source/medium.php:12 Stack trace: #0 /var/www/html/DVWA/vulnerabilities/sqli/source/medium.php(12): mysqli_query() #1 /var/www/html/DVWA/vulnerabilities/sqli/index.php(34): require_once('...') #2 {main} thrown in **/var/www/html/DVWA/vulnerabilities/sqli/source/medium.php** on line

**12**″

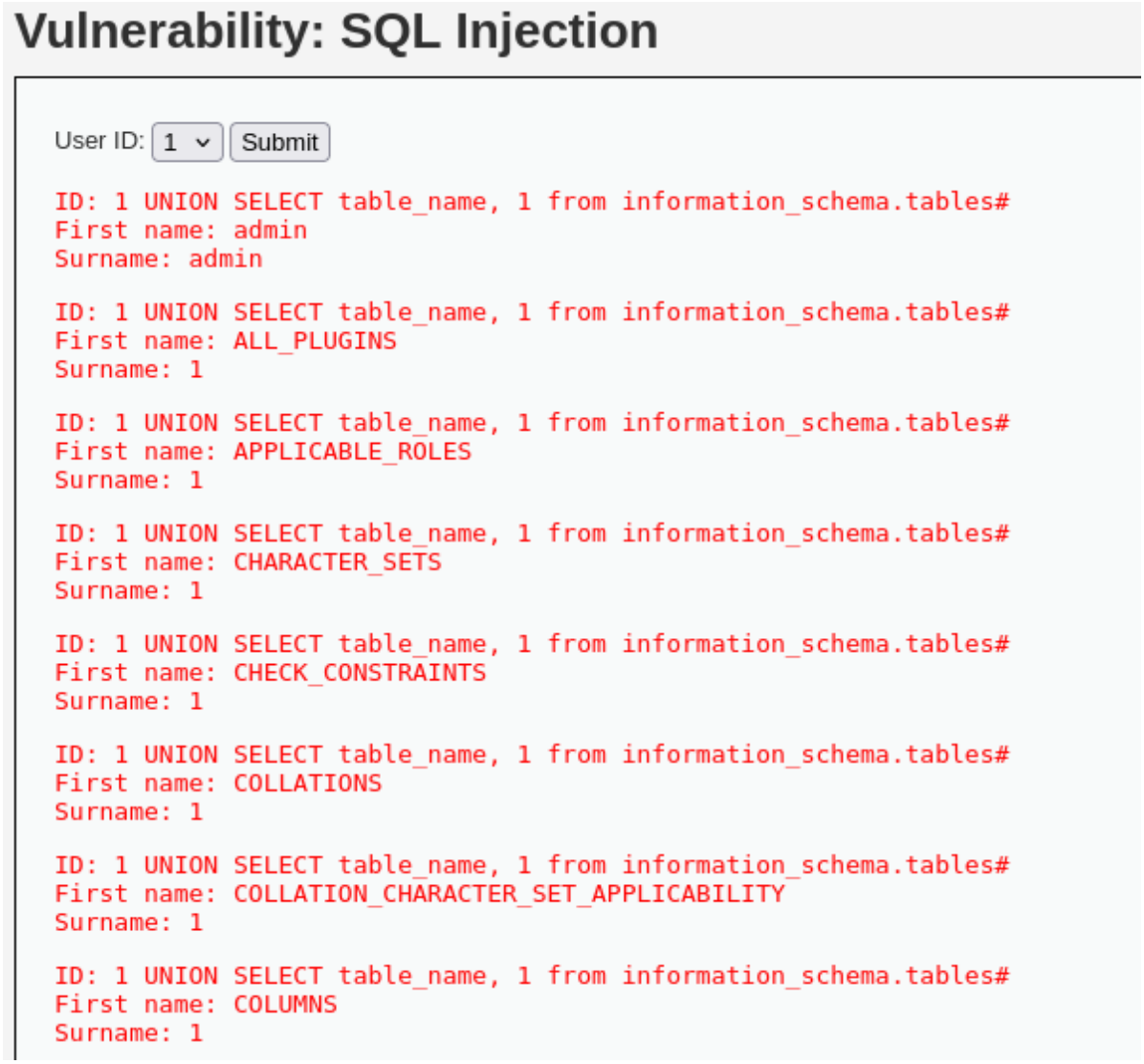
값을 ″2′ or ′1′=′1″로 수정하여 다시 제출해보면 아래와 같은 오류가 발생한다.

- **Fatal error**: Uncaught mysqli_sql_exception: You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '₩' or ₩'1₩'=₩'1' at line 1 in /var/www/html/DVWA/vulnerabilities/sqli/source/medium.php:12 Stack trace: #0 /var/www/html/DVWA/vulnerabilities/sqli/source/medium.php(12): mysqli_query() #1 /var/www/html/DVWA/vulnerabilities/sqli/index.php(34): require_once('...') #2 {main} thrown in **/var/www/html/DVWA/vulnerabilities/sqli/source/medium.php** on line **12**

밑줄 친 부분을 확인해보면 입력한 값에 이스케이핑 처리가 되어있는 것을 확인할 수 있다. 따라서 Low 레벨에서 공격에 성공했던 구문은 더 이상 유효하지 않다. 하지만 이스케이핑되지 않는 특수문자나 구문을 사용한다면 여전히 SQL Injection 공격이 가능하다.

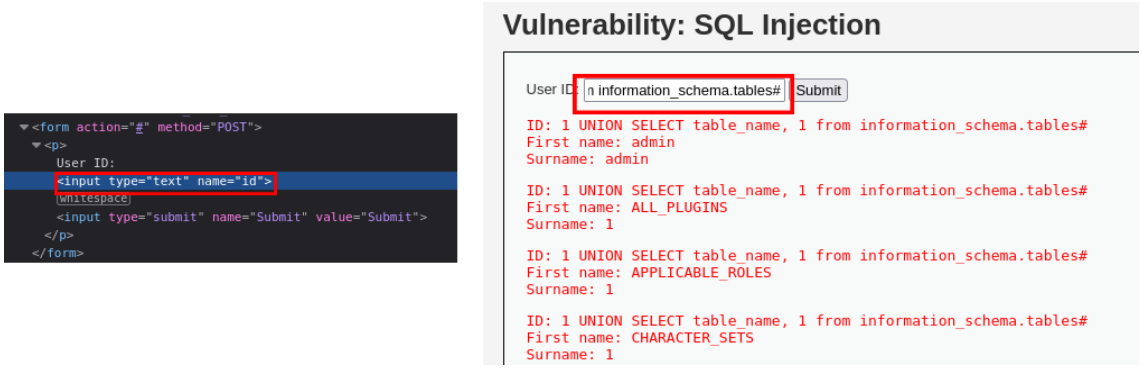



[사진 7] 드롭다운 메뉴 옵션 값 수정




[사진 8] SQL Injection 결과

"1 UNION SELECT table_name, 1 from information_schema.tables#"을 로 값을 수정하고 제출해보면 위의 사진처럼 SQL Injection이 성공하여 DB의 모든 테이블 이름이 출력되는 것을 확인할 수 있다.




[사진 9] 드롭다운 메뉴 텍스트 입력창으로 수정

추가적으로 DOM을 수정하여 드롭다운 메뉴를 텍스트 입력창으로 바꾸어 시도하는 것도 가능하다.

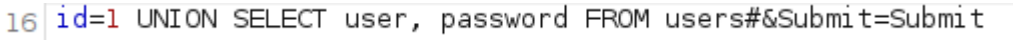
### 2) Ruquest 조작

Proxy는 BurpSuite를 사용하여 실습을 진행했다.

BurpSuite의 Proxy>Intercept 설정을 On으로 설정한 후 DVWA SQL Injection 페이지에서 UserID를 1로 설정하고 제출버튼을 누른다.

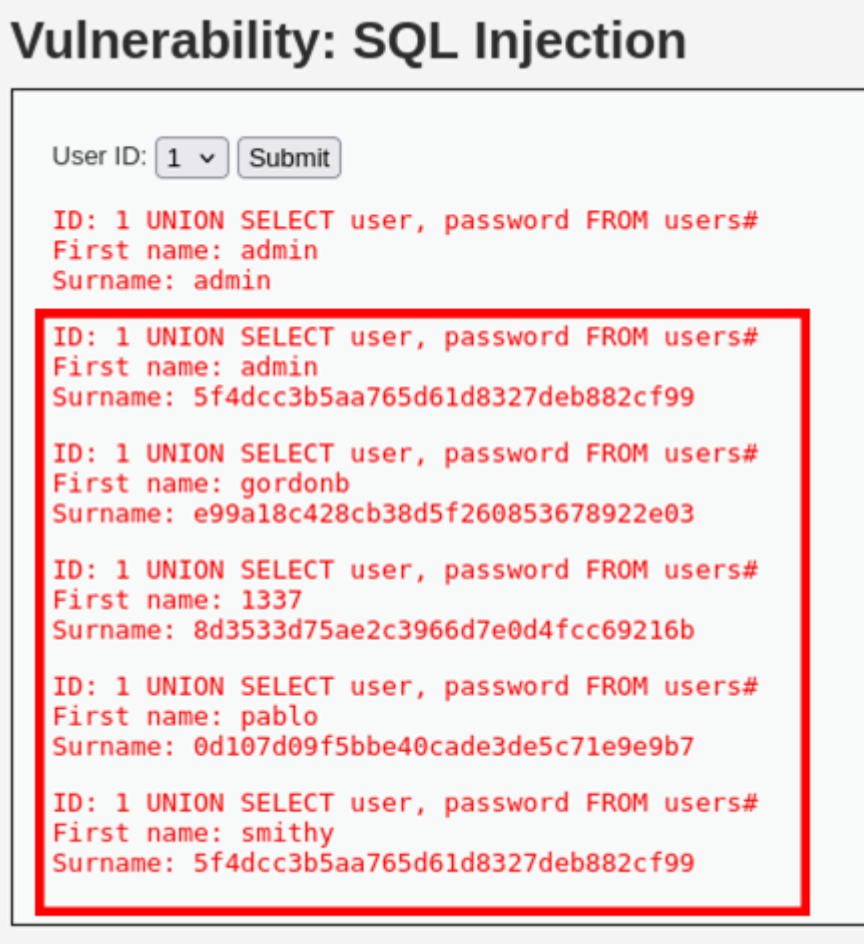

[사진 10] 캡처한 Request 패킷

위처럼 BurpSuite의 Request 패킷이 잡힌 것과 파라미터인 id값이 1로 표시된 것을 확인할 수 있다.




[사진 11] 수정된 파라미터

id의 값을 위처럼 수정한 후 패킷을 Forward 해준다.

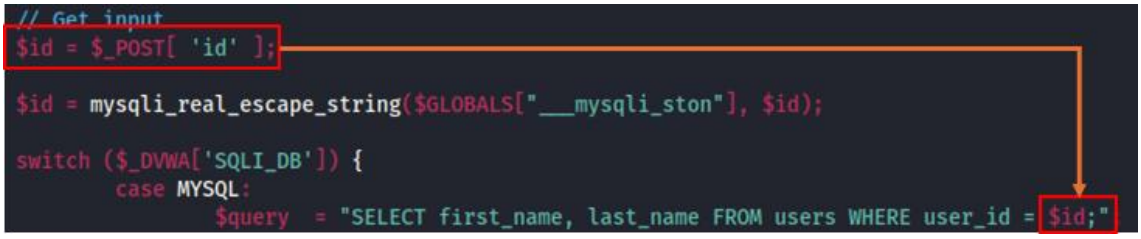



[사진 12] SQL Injection 결과

그러면 SQL Injection이 성공하여 유저 이름과 비밀번호가 출력되는 것을 확인할 수 있다.

## 2.3 SQL Injection 대응방안

1. Prepared Statement를 사용한다.




[사진 13] 기존 쿼리문 생성 코드

기존 코드에서는 사용자의 입력에 간단한 sql 이스케이프 후 SQL 구분에 삽입했었다. 하지만 이는 SQL Injection 취약점을 야기할 수 있다.

```php
// 사용자 입력 받기
$id = $_POST;

// DB 연결
$mysqli = $GLOBALS;

// Prepared Statement 생성
$stmt = $mysqli→prepare("SELECT first_name, last_name FROM users WHERE
user_id = ?");

// 바인드 및 실행
$stmt→bind_param('i', $id);
```

[사진 14] Prepared Statement 적용 쿼리문 생성 코드

위의 예시코드처럼 Prepared Statement 를 사용하게 되면 사용자 입력이 미리 준비된 구문에 바인딩될 때 형변환이 일어나면서 공격자가 의도한 코드는 실행되지 않을 것이다.

2. 사용자 입력값을 Regex 나 화이트리스트 기반으로 필터링한다.

```php
// 화이트리스트
$allowed_ids = [1, 2, 3, 4, 5];

$id = $_POST[ 'id' ];

$raw_id = trim((string)$_POST['id']);


if (!ctype_digit($raw_id)) {
    http_response_code(400);
    echo 'Invalid id format.';
    exit;
}

$int_id = (int)$raw_id;

// 화이트리스트 검사
if (!in_array($int_id, $allowed_ids, true)) {
    http_response_code(400);
    echo 'Invalid id.';
    exit;

}
```

[사진 15] 화이트리스트 기반 사용자 입력 필터링 코드

본 실습 페이지에서는 사용자 입력값을 드롭다운 메뉴로 1, 2, 3, 4, 5 로 제한해두고 있었다. 따라서 위 예시코드처럼 화이트리스트를 사용하여 필터링하면 클라이언트에서 다른 값으로 조작하여 입력해도 1, 2, 3, 4, 5 와 일치하지 않는다면 필터링될 것이다.

3. DB 계정에 필요한 최소한의 권한만 부여한다.

본 실습 페이지에서는 UserID 와 매칭되는 이름을 출력하는 것이었다. 하지만 실습에서 확인한 것처럼 전체 DB 의 정보, 다른 테이블의 데이터 조회 등 모든 데이터에 접근할 수 있었다.

DB 계정에 필요한 최소한의 권한만 부여하여 불필요한 읽기, 쓰기 권한을 제한하면 한쪽에서 취약점이 발생하여 계정 권한이 탈취되더라도 피해를 최소화할 수 있을 것이다.

# 3. Blind SQL Injection

## 3.1 개요

Blind SQL Injection은 공격자가 애플리케이션을 통해 데이터베이스 쿼리 취약점을 악용하지만, 악성 쿼리의 직접적인 결과를 볼 수 없는 공격입니다.

Blind SQL Injection 공격은 일반적인 SQL Injection과 달리 공격자는 데이터가 웹 페이지에 직접 표시되거나 데이터베이스에서 자세한 오류 메시지를 확인할 수 없습니다. 대신, 공격자는 데이터베이스에 일련의 '예/아니오(True/False)' 쿼리를 하고, 명확한 오류나 경보를 발생시키지 않고 간접적인 피드백을 통해 정보를 추출합니다.

Blind SQL Injection은 일반적인 SQL Injection보다 느리지만 민감한 데이터를 탈취할 수 있으니 주의해야 합니다.

### 3.1.1 Blind SQL Injection 유형

**Boolean-Based Blind SQLi(SQL Injection)**

공격자는 참/거짓 조건을 강제하는 쿼리를 보내고, 애플리케이션이 어떻게 응답하는지 관찰합니다.

이를 통해 공격자는 관리자 계정의 비밀번호 첫 글자가 'A'인지 'B'인지 등을 추론하며 데이터를 추출할 수 있습니다.

**Time-Based Blind SQLi**

공격자는 응답 지연을 이용하여 데이터를 추출합니다. 조건이 참일 때 데이터베이스 응

답을 의도적으로 지연시키도록 SQL 페이로드를 작성하고 애플리케이션이 응답하는 데 걸리는 시간을 측정함으로써, 조건이 참인지 거짓인지를 파악합니다.

## 3.2 Blind SQL Injection 실습


[사진 16] DVWA SQL Injection (Blind) 페이지

DVWA 실습 페이지의 왼쪽 탭에서 'SQL Injection (Blind)' 탭으로 이동하면 SQL Injection 페이지와 마찬가지로 User ID를 선택할 수 있는 드롭다운 메뉴와 'Submit' 버튼이 보인다.


[사진 17] 드롭다운 메뉴와 양식 제출 결과

드롭다운 메뉴를 확인해보면 정수 1~5까지의 범위의 숫자를 선택할 수 있고 숫자를 선택하고 제출버튼을 누르면 ID에 해당하는 정보가 데이터베이스에 존재하는지 유무를 하

단에 출력한다.

DVWA의 SQLi 실습과 동일하게 Low 레벨에 있었던 텍스트 입력창이 드롭다운 메뉴로 변경되었다.

```php
<?php

if( isset( $_POST[ 'Submit' ]  ) ) {
	// Get input
	$id = $_POST[ 'id' ];
	$exists = false;

	switch ($_DVWA['SQLI_DB']) {
		case MYSQL:
			$id = ((isset($GLOBALS["___mysqli_ston"]) && is_object($GLOBALS["___mysqli_ston"])) ? mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $id ) : ((trigger_error("[MySQLConverterTool] Fix the mysql_escape_string() call! This code does not work.", E_USER_ERROR)) ? "" : ""));

			// Check database
			$query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
			try {
				$result = mysqli_query($GLOBALS["___mysqli_ston"],  $query ); // Removed 'or die' to suppress mysql errors
			} catch (Exception $e) {
				print "There was an error.";
				exit;
			}

			$exists = false;
			if ($result !== false) {
				try {
					$exists = (mysqli_num_rows( $result ) > 0); // The '@' character suppresses errors
				} catch(Exception $e) {
					$exists = false;
				}
			}

			break;
		case SQLITE:
			global $sqlite_db_connection;

			$query  = "SELECT first_name, last_name FROM users WHERE user_id = $id;";
			try {
				$results = $sqlite_db_connection->query($query);
				$row = $results->fetchArray();
				$exists = $row !== false;
			} catch(Exception $e) {
				$exists = false;
			}
			break;
	}

	if ($exists) {
		// Feedback for end user
		$html .= '<pre>User ID exists in the database.</pre>';
	} else {
		// Feedback for end user
		$html .= '<pre>User ID is MISSING from the database.</pre>';
	}
}

?>
```

[사진 18] DVWA SQL Injection Medium 소스코드

Low 난이도와 마찬가지로 사용자 입력값을 특수문자 이스케이핑을 제외한 어떤 필터링도 하지 않고 바로 SQL 구문에 삽입한다. 이는 Blind SQL Injection 취약점을 유발한다.

또한 쿼리의 결과가 1개 이상이라면 $exists 값을 참으로 설정하고 1개 미만이라면 False

로 설정한 후 $exists의 값에 따라 하단에 결과를 나타내는 문자열을 출력한다.

### 3.2.1 시나리오

소스코드를 통해 확인한 결과 사용자 입력 필터링을 하지 않기 때문에 SQLi 실습에서 했던 것처럼 클라이언트 드롭다운 메뉴를 우회하기 위해서 DOM 수정, Proxy를 이용한 Request 패킷 수정을 통해 Blind SQLi가 가능하다.

#### 1) DOM 수정

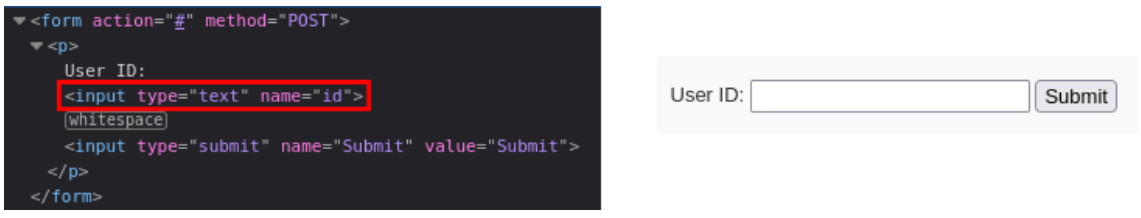
1. DOM을 조작하여 드롭다운 메뉴의 값을 수정한다.
2. 수정한 값을 선택하고 제출한다.
3. 서버는 수정한 값을 그대로 SQL 구문에 삽입한다.

#### 2) Request 수정

1. 드롭다운 메뉴에서 아무 값이나 선택한 후 제출하여 서버에 요청을 보낸다.
2. Proxy를 이용하여 클라이언트의 요청을 잡은 후 'id'값을 수정한다.
3. 서버는 공격자가 수정한 값을 그대로 SQL 구문에 삽입한다

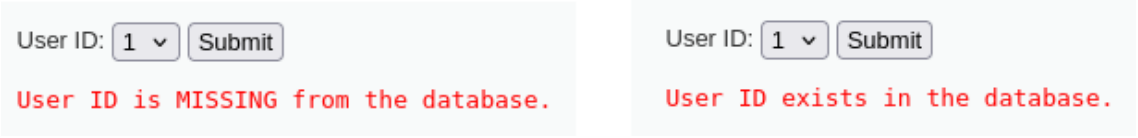
### 3.2.2 실습

#### 1) DOM 수정



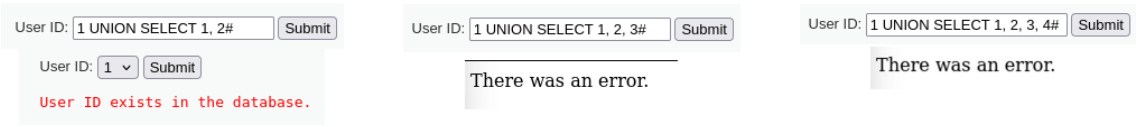

[사진 19] 드롭다운 메뉴 텍스트 입력창으로 수정

브라우저의 개발자 도구를 통해 DOM을 수정하여 드롭다운 메뉴를 텍스트 입력창으로 변경했다.



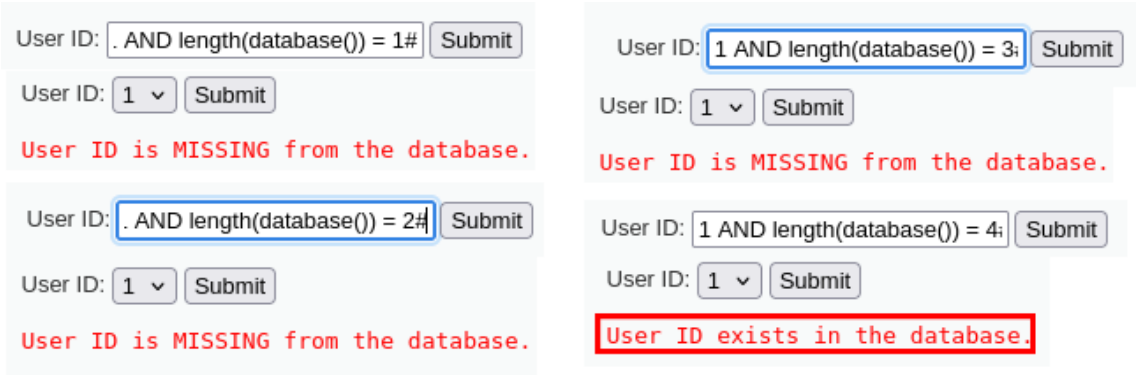

[사진 20] 0입력 결과(왼쪽), 1입력 결과(오른쪽)

입력창을 통해 0, 1을 순서대로 입력해보았다. 0을 입력했을 때에는 False 구문이, 1을 입력했을 때에는 True 구문이 출력되었다.




[사진 21] Column 수 추론 Blind SQLi

Column의 수를 알아내기 위해 '1 UNION SELECT [1, 2, ...]#'을 입력했다. 그 결과 '1, 2'일 때에만 오류가 나지 않고 True 구문이 출력되는 것으로 보아 현재 쿼리문의 컬럼수가 2개임을 알 수 있다.




[사진 22] DB이름의 길이 추론 Blind SQLi

DB 이름의 길이를 알아내기 위해 '1 AND length(database()) = [숫자]#'을 입력했다. 그 결과 4일 때에만 True 문자열이 출력되는 것으로 보아 현재 데이터베이스 이름의 길이는 4인 것을 알 수 있다.

### 2) Request 수정


[사진 23] 캡처한 Request 패킷

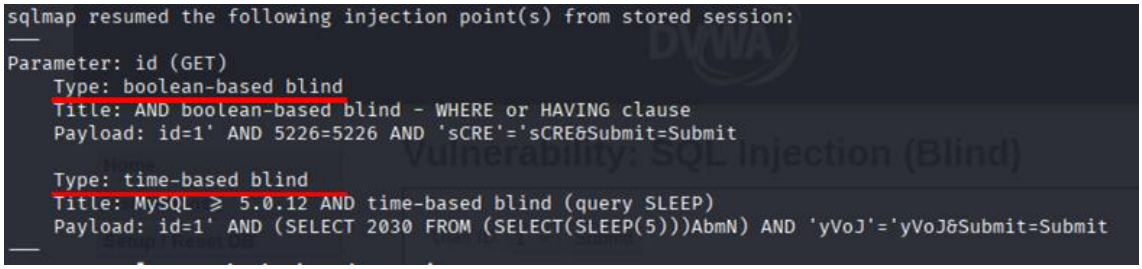
BurpSuite의 Intercept 기능을 실행하고 드롭다운 버튼으로 ID값을 설정하고 제출한다. 위 사진처럼 Request 패킷이 잡힌 것을 확인할 수 있다. id값을 바꾸면 1)과 같은 방법으로 Blind SQL Injection이 가능하다. 하지만 시간이 너무 오래 걸리기에 자동화 도구를 사용할 것이다.

인젝션 자동화/점검 도구인 SQLmap을 사용한다.


[사진 24] SQLmap id 파라미터 취약점 분석

BurpSuite에서 쿠키값과 URL을 복사하여 아래의 명령어에 입력하여 터미널에서 실행시켜준다.

- sqlmap --cookie [쿠키값] -u [URL] -p id



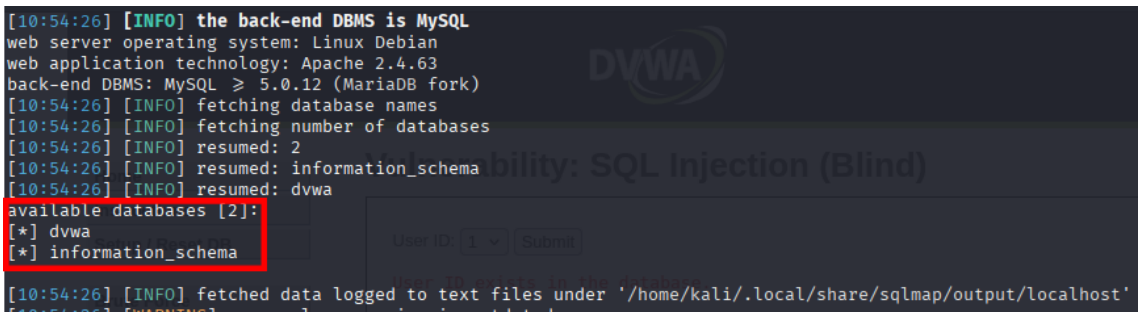

[사진 25] id 파라미터 취약점 분석결과

실행 결과 id 파라미터에는 Boolean-based Blind SQLi 취약점과 Time-based Blind SQLi 취약점이 존재하는 것을 알 수 있다.


[사진 26] SQLmap Blind SQLi dbs목록 분석

아래의 명령어를 통해 접근가능한 DB 목록을 확인할 수 있다.

- sqlmap --cookie [쿠키값] -u [URL] -p id --dbs




[사진 27] SQLmap dbs 목록 분석결과

실행 결과 접근 가능한 DB는 dvwa, information_schema가 존재하는 것을 알 수 있다.

SQLmap의 다른 명령어 들을 활용하면 더욱 많은 정보를 확인할 수 있다.

소스코드에서 확인할 수 있듯이 id 파라미터 값을 필터링 없이 SQL 구문에 삽입하기 때문에 바로 서버에 요청을 보낼 수 있는 자동화 툴을 이용해서도 실습할 수 있다.

## 3.3 Blind SQL Injection 대응방안

기존 SQL Injection의 대응방안인 Prepared Statement, 사용자 입력값 검증, DB 계정 권한 최소화 등은 동일하게 적용될 수 있다.

1. 쿼리의 참/거짓을 구분할 수 있는 정보(에러 메시지, 패킷의 길이 등) 최소한으로 제공한다. (Boolean-based Blind SQLi)

```php
if ($exists) {
        // Feedback for end user
        $html .= '<pre>User ID exists in the database.</pre>';
} else {
        // Feedback for end user
        $html .= '<pre>User ID is MISSING from the database.</pre>';
}
```

[사진 28] 기존 결과 메시지 출력 코드

기존의 코드에서는 쿼리 결과의 유무에 따라 다른 메시지가 출력되어 공격자가 데이터를 추출할 가능성이 있었다.

```php
// 기본 응답 메시지
$response_message = '<pre>Invalid request or user not found.</pre>';

// Feedback for end user - 항상 동일한 메시지를 제공
$html .= $response_message;
```

[사진 29] 동일한 결과 메시지 출력 코드

위의 예시코드처럼 기본 응답 메시지를 설정하고 항상 동일한 패턴의 메시지를 제공한다면 공격자가 피드백을 이용하여 데이터를 추측하는 것을 방지할 수 있다. 하지만 UX가 나빠질 수 있으므로 이 방법은 사용에 주의해야 한다.

2. 쿼리 타임아웃을 설정한다. (Time-based Blind SQLi)

기존 코드에서는 쿼리 타임아웃을 설정하지 않았었다. 만약 공격자가 Time-based Blind SQLi 공격을 시도하려고 했었다면 응답 시간의 차이가 명확하게 나기 때문에 공격자가 분석하기 쉬웠었다.

```php
// 쿼리 타임아웃 설정 (예: 5초)
if (defined('MYSQLI_OPT_READ_TIMEOUT')) {
    $mysqli->options(MYSQLI_OPT_READ_TIMEOUT, 5);
}
// 연결 타임아웃 설정 (예: 10초) ((1))
$mysqli->options(MYSQLI_OPT_CONNECT_TIMEOUT, 10);
```

[사진 30] 쿼리 타임아웃 설정 코드

위의 예시코드처럼 시간 제한을 5초로 설정하는 등의 쿼리, 연결 시간의 제한 시간을 설정해놓는다면 공격자가 데이터를 추측하는 것을 어렵게 할 수 있다.

# 4. 결론

## 4.1 정리

| 공격 벡터 | SQLi | Blind SQLi |
|---|---|---|
| 공격 표면 | 악성 SQL 코드를 주입할 수 있는 웹 애플리케이션의 모든 지점 (사용자 입력 필드, HTTP헤더 등) | SQLi와 동일 |
| 주요 피해 | DB 데이터 탈취, DB조작 | SQLi와 동일 |
| 대응 방안 | - Prepared Statement<br>- 사용자 입력값 검증<br>- DB계정 권한 최소화 | - SQLi 대응방안과 동일<br>+ 쿼리 참/거짓 구분 정보 노출 최소화<br>+ 쿼리 타임아웃 설정 |