New Techniques in SQLi Obfuscation

SQL never before used in SQL Injection

Nick Galbreath
@ngalbreath
nick@client9.com

DEFCON 20 at the Rio in sunny Las Vegas
2012-07-27 Friday 4:20 pm!
Follow along or get the latest version:

http://slidesha.re/MfOiNR

That's an OH, not a zero
SQL Specification

- 625 pages of plain text
- 119 pages of pure BNF
- No one implements exactly
- Everyone has extensions, exceptions, bugs
Some of the regular expressions used by PHPIDS 0.7
Analyzing SQL and SQLi

• Libinjection is a Quasi-SQL tokenizer
• [https://github.com/client9/libinjection](https://github.com/client9/libinjection)
• Tries to handle all vendor special cases
• Run all SQLi through it, see what code paths in the parser aren't being triggered
• *(note, libinjection is a work in progress, biased toward MySQL, PgSQL for the moment)*
Sources

Tens of thousands attacks of varying quality

- Output from SQLi vulnerability scanners against dummy sites
- Published attacks
- HOW-TO guides
- Stuff we see at Etsy
Lots of Dark Corners

• We'll review many of the SQL oddities that aren't actively being used or are interesting enough to re-review.

• Great for new fuzzers, vulnerability scanners, WAF builders and validators.
NULL
MySQL NULL Alias

MySQL NULL can written as \N
case sensitive. \n is not a null.

This means any WAF that does a "to_lower" on the user input and looks for "null" will miss this case.
NULL PGSQL

- ISNULL, NOTNULL (same as IS NULL), this is a function in MSSQL
- "IS [NOT] UNKNOWN"
- "IS [NOT] DISTINCT"
Numbers
Floating Point

- digits
- digits[.]
- digits[.]digits
- digits[eE]digits
- digits[eE][+]digits
- digits[.]digits[eE]digits

Optional starts with [+] 
Optional ending with [dDfF] (Oracle)
Exceptions

• 1.AND 2 (no space between "1." "AND") some parsers accept, some don't

• lel vs. lel.0 ?
Oracle Special Literals

numbers without numbers!

- binary_double_infinity
- binary_double_nan
- binary_float_infinity
- binary_float_nan

might be case sensitive
Hexadecimal Literals

- 0xDEADbeef  MySQL, MSSQL
  0x is case sensitive
- 0x  (empty string) MSSQL only
- x'DEADbeef'  PgSQL
Binary Literals

- `b'10101010'` MySQL, PgSQL
- `0b010101` MySQL
- case sensitive
Money Literals

- MSSQL has a money type.
- -$45.12
- $123.0
- +$1,000,000.00 Commas ignored
- Haven't really experiments with this yet.
- Does it auto-cast to a float or int type?
Comments
MySQL # Comment

- '#' signals an till-end-of-line Comment
- Well used in SQLi attacks
- However... '#' is an operator in PgSQL. Beware that s/\#.\n// will delete code that needs inspecting.
- Lots of other MySQL comment oddities: http://dev.mysql.com/doc/refman/5.6/en/comments.html
PGSQL Comments

• Besides the usual -- comment
• PgSQL has recursive C-Style Comments
  /* foo /* bar */ */
• Careful! What happens when you 'remove comments' in /* */ */ UNION ALL /* */ */
Strings
C-Style String Merging

• C-Style consecutive strings are merged into one.

• SELECT 'foo' 'bar';

• SELECT 'foo' "bar"; (mysql)

• SQL Spec and PgSQL requires a newline between literals:
  SELECT 'foo'
    'bar';
Standard Unicode

- N'....' or n'...
- MSSQL Case-sensitive 'N'
- Not sure on escaping rules.
MySQL Ad-Hoc Charset

- `charset`..
- `latin1`..
- `utf8`..
A dollar-quoted string constant consists of a dollar sign ($), an optional "tag" of zero or more characters, another dollar sign, an arbitrary sequence of characters that makes up the string content, a dollar sign, the same tag that began this dollar quote, and a dollar sign. For example, here are two different ways to specify the string "Dianne's horse" using dollar quoting:

$$Dianne's horse$$
$SomeTag$Dianne's horse$SomeTag$

What more fun? They can be nested!
...

... This variant starts with \( \texttt{U\&} \) (upper or lower case \( \texttt{U} \) followed by ampersand) immediately before the opening double quote, without any spaces in between, for example \( \texttt{U\&"f\text{o}\text{o}"} \). (Note that this creates an ambiguity with the operator \( \texttt{&} \). Use spaces around the operator to avoid this problem.) Inside the quotes, Unicode characters can be specified in escaped form by writing a backslash followed by the four-digit hexadecimal code point number or alternatively a backslash followed by a plus sign followed by a six-digit hexadecimal code point number. For example, the identifier "data" could be written as

\( \texttt{U\&"d\text{"0061\"t\text{"000061}"} } \)

The following less trivial example writes the Russian word "slon" (elephant) in Cyrillic letters:

\( \texttt{U\"\text{"0441\"043B\"043E\"043D}" } \)

If a different escape character than backslash is desired, it can be specified using the \texttt{UESCAPE} clause after the string, for example:

\( \texttt{U\"d!0061t!+000061" UESCAPE '!' } \)
Oracle Q String

http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/fundamentals.htm#autoId6

q'!...!' notation allows use of single quotes inside literal

string_var := q'!I'm a string!';

You can use delimiters [, {, <, and (, pair them with ], }, >, and ), pass a string literal representing a SQL statement to a subprogram, without doubling the quotation marks around 'INVALID' as follows:

func_call(q'[SELECT index_name FROM user_indexes
   WHERE status = 'INVALID']');
Operators and Expressions
Operators!

- ! and !! Factorial (pgsql)
- |/ square root (pgsql)
- ||/ cube root (pgsql)
- # bitwise XOR (pgsql, conflicts with MySQL)
- ** exponents (oracle)
More Operators!

- `!=, <=> (mysql), <> (mssql), ^= (oracle)`
- `!>, !<` not less than, (mssql)
- `\ Bitwise XOR (oracle)"
Expressions!

- Using the common query extension of "OR 1=1"
- Besides using literals, one can use functions:
  - \( \cos(0) = \sin(\pi/2) \)
  - \( \cos(@\text{VERSION}) = -\sin(@\text{VERSION} + \pi/2) \)
EXCEPT (mssql)
MINUS (Oracle)

• Like UNION, UNION ALL

• But returns all results from first query minus/except the ones from the second query

• There is also INTERSECT as well.

• I think someone clever could use these, typically not in WAF rules.
Side Note: "IN" lists

• e.g. ....WHERE id IN (1,2,3,4) ....

• These have to be manually created.

• There is no API or parameter binding for this construct in any platform, framework or language.

• There is no consistent, safe way to make this (other than convention, validation)
Why don't we see more attacks using these techniques?

• Dumb attacks work (for now)
• I don't get see the more advanced attacks
What's Next?

• Add more parsing rules to libinjection
• More testing frameworks
• Investigate BIGINT types
• pgsql has a regexp engine, and various other datatypes
• Worry about various character encodings
Primary References


• http://www.postgresql.org/docs/9.1/static/functions.html


• http://docs.oracle.com/cd/B28359_01/
Thanks!

Nick Galbreath
@ngalbreath
nickg@client9.com

https://github.com/client9/libinjection