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 zer0

SQL Specification

- http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt
- 625 pages of plain text
- http://savage.net.au/SQL/sql-2003-2.bnf
- 119 pages of pure BNF
- No one implements exactly
- Everyone has extensions, exceptions, bugs

Regexp Based WAF

```
(?:\)\s*when\s*\d+\s*then)|(?:"\s*(?:#|--|{))|(?:\/\*!\s?\d+)|(?:ch(?:a)?r\s*\(\s*\d)|(?:(?:(n?and|x?or|not)\s+|\|\||\&\&)\s*\w+\()
(?:[\s()]case\s*\()|(?:\)\s*like\s*\()|(?:having\s*[^\s]+\s*[^\w\s])|(?:if\s?\([\d\w]\s*[=<>~])
(?:"\s*or\s*"?\d)|(?:\\x(?:23|27|3d))|(?:^.?"$)|(?:(?:^["\\]*(?:[\d"]+|[^"]+"))+\s*(?:n?and|x?or|not|\|\||\&\&)\s*[\w"[+&!@(),.-])|(?:[^\w\s]\w+\s*[|-]
\s*"\s*\w)|(?:@\w+\s+(and|or)\s*["\d]+)|(?:@[\w-]+\s(and|or)\s*[^\w\s])|(?:[^\w\s:]\s*\d\W+[^\w\s]\s*".)|(?:\Winformation_schema|table_name\W)
(?:"\s*\*.+(?:or|id)\W*"\d)|(?:\^")|(?:^[\w\s"-]+(?<=and\s)(?<=or\s)(?<=xor\s)(?<=nand\s)(?<=not\s)(?<=\|\|)(?<=\&\&)\w+\()|(?:"[\s\d]*[^\w\s]+\W*\d
\W*.*["\d])|(?:"\s*[^\w\s?]+\s*[^\w\s]+\s*")|(?:"\s*[^\w\s]+\s*[\W\d].*(?:#|--))|(?:".*\*\s*\d)|(?:"\s*or\s[^\d]+[\w-]+.*\d)|(?:[()*<>%+-][\w-]+[^\w\s]
+"[^,])
(?:\d"\s+"\s+\d)|(?:^admin\s*"|(\/\*)+"+\s?(?:--|#|\/\*|{)?)|(?:"\s*or[\w\s-]+\s*[+<>=(),-]\s*[\d"])|(?:"\s*[^\w\s]?=\s*")|(?:"\W*[+=]+\W*")|(?:"\s*[!=|]
[\d\s!=+-]+.*["(].*$)|(?:"\s*[!=|][\d\s!=]+.*\d+$)|(?:"\s*like\W+[\w"(])|(?:\sis\s*0\W)|(?:where\s[\s\w\.,-]+\s=)|(?:"[<>~]+")
(?:union\s*(?:all|distinct|[(!@]*)?\s*[([]*\s*select)|(?:\w+\s+like\s+\")|(?:like\s*"\%)|(?:"\s*like\W*["\d])|(?:"\s*(?:n?and|x?or|not |\|\||\&\&)\s+[\s
\w]+=\s*\w+\s*having)|(?:"\s*\*\s*\w+\W+")|(?:"\s*[^?\w\s=.,;)(]+\s*[(@"]*\s*\w+\W+\w)|(?:select\s*[\[\]()\s\w\.,"-]+from)|(?:find_in_set\s*\()
(?:in\s*\(+\s*select)|(?:(?:n?and|x?or|not |\|\||\&\&)\s+[\s\w+]+(?:regexp\s*\(|sounds\s+like\s*"|[=\d]+x))|("\s*\d\s*(?:--|#))|(?:"[%&<>^=]+\d\s*(=|
or))|(?:"\W+[\w+-]+\s*=\s*\d\W+")|(?:"\s*is\s*\d.+"?\w)|(?:"\|?[\w-]{3,}[^\w\s.,]+")|(?:"\s*is\s*[\d.]+\s*\W.*")
(?:[\d\W]\s+as\s*["\w]+\s*from)|(?:^[\W\d]+\s*(?:union|select|create|rename|truncate|load|alter|delete|update|insert|desc))|(?:(?:select|create|rename|
truncate|load|alter|delete|update|insert|desc)\s+(?:(?:group_)concat|char|load_file)\s?\(?)|(?:end\s*\);)|("\s+regexp\W)|(?:[\s(]load_file\s*\()
(?:@.+=\s*\(\s*select)|(?:\d+\s*or\s*\d+\s*[\-+])|(?:\/\w+;?\s+(?:having|and|or|select)\W)|(?:\d\s+group\s+by.+\()|(?:(?:;|#|--)\s*(?:drop|alter))|(?:
(?:;|#|--)\s*(?:update|insert)\s*\w{2,})|(?:[^\w]SET\s*@\w+)|(?:(?:n?and|x?or|not |\|\||\&\&)[\s(]+\w+[\s)]*[!=+]+[\s\d]*["=()])
(?:"\s+and\s*=\W)|(?:\(\s*select\s*\w+\s*\()|(?:\*\/from)|(?:\+\s*\d+\s*\+\s*@)|(?:\w"\s*(?:[-+=|@]+\s*)+[\d(])|(?:coalesce\s*\(|@@\w+\s*[^\w\s])|(?:\W!
+"\w)|(?:";\s*(?:if|while|begin))|(?:"[\s\d]+=\s*\d)|(?:order\s+by\s+if\w*\s*\()|(?:[\s(]+case\d*\W.+[tw]hen[\s(])
(?:(select|;)\s+(?:benchmark|if|sleep)\s*?\(\s*\(?\s*\w+)
(?:create\s+function\s+\w+\s+returns)|(?:;\s*(?:select|create|rename|truncate|load|alter|delete|update|insert|desc)\s*[\[(]?\w{2,})
(?:alter\s*\w+.*character\s+set\s+\w+)|(";\s*waitfor\s+time\s+")|(?:";.*:\s*goto)
(?:procedure\s+analyse\s*\()|(?:;\s*(declare|open)\s+[\w-]+)|(?:create\s+(procedure|function)\s*\w+\s*\(\s*\)\s*-)|(?:declare[^\w]+[@#]\s*\w+)|(exec\s*\
(\s*@)
(?:select\s*pg_sleep)|(?:waitfor\s*delay\s?"+\s?\d)|(?:;\s*shutdown\s*(?:;|--|#|\/\*|{))
(?:\sexec\s+xp_cmdshell)|(?:"\s*!\s*["\w])|(?:from\W+information_schema\W)|(?:(?:(?:current_)?user|database|schema|connection_id)\s*\([^\)]*)|(?:";?
\s*(?:select|union|having)\s*[^\s])|(?:\wiif\s*\()|(?:exec\s+master\.)|(?:union select @)|(?:union[\w(\s]*select)|(?:select.*\w?user\()|(?:into[\s+]+
(?:dump|out)file\s*")
(?:merge.*using\s*\()|(execute\s*immediate\s*")|(?:\W+\d*\s*having\s*[^\s\-])|(?:match\s*[\w(),+-]+\s*against\s*\()
(?:,.*[)\da-f"]"(?:".*"|\Z|[^"]+))|(?:\Wselect.+\W*from)|((?:select|create|rename|truncate|load|alter|delete|update|insert|desc)\s*\(\s*space\s*\()
(?:\[\$(?:ne|eq|lte?|gte?|n?in|mod|all|size|exists|type|slice|or)\])
(?:(sleep\((\s*)(\d*)(\s*)\)|benchmark\((.*)\,(.*)\)))
(?:(union(.*)select(.*)from))
(?:^(-0000023456|4294967295|4294967296|2147483648|2147483647|0000012345|-2147483648|-2147483649|0000023456|2.2250738585072007e-308|1e309)$)
```

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
- 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[.][eE]digits
- digits[.]digits[eE]digits
- digits[.]digits[eE][+-]digits
- [.]digits
- [.]digits[eE]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
- 1e1 vs. 1e1.0 ?

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'....'

PGSQL Dollar Quoting

From http://www.postgresql.org/docs/9.1/static/sql-syntax-lexical.html#SQL-SYNTAX-COMMENTS

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!

PGSQL Unicode

From http://www.postgresql.org/docs/9.1/static/sql-syntax-lexical.html emphasis mine:

... This variant starts with `U&` (upper or lower case U followed by ampersand) immediately before the opening double quote, without any spaces in between, for example `U&"foo"`. (Note that this creates an ambiguity with the operator `&`. 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**

`U&"d\0061t\+000061"`

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

```
U&"\0441\043B\043E\043D"
```

If a **different escape character** than backslash is desired, it can be **specified** using the `UESCAPE` clause **after the string**, for example:

```
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(@VERSION) = -SIN(@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://dev.mysql.com/doc/refman/5.6/en/func-op-summary-ref.html
- http://www.postgresql.org/docs/9.1/static/functions.html
- http://msdn.microsoft.com/en-us/library/bb510741
- http://docs.oracle.com/cd/B28359_01/

Thanks!

Nick Galbreath

@ngalbreath

nickg@client9.com

https://github.com/client9/libinjection