Square Peg, Round Hole

Automatically stopping SQL injection by matching shapes

TQ Hirsch <thequux@thequux.com>

2015-05-30T19:00:00+0200

What you just saw

- Works.

What you just saw

- Works.
- But using it is very manual.

What you just saw

- Works.
- But using it is very manual.
- It's pretty C++.

What you just saw

- Works.
- But using it is very manual.
- It's pretty C++.
- Modifying the filter involves code generation

What you're about to see

- Not perfect

What you're about to see

- Not perfect
- Won't work for every app

What you're about to see

- Not perfect
- Won't work for every app
- Changes to your app are minimal

What you're about to see

- Not perfect
- Won't work for every app
- Changes to your app are minimal
- So were the changes to PostgreSQL

```
db=# select * from users where name = 'bob'
                              and password = 'p4ssw0rd';
  name   |           password
---------+-------------------------------
 bob     | p4ssw0rd
(1 row)
```

```
db=# select * from users where name = 'bob'
                                and password = 'p4ssw0rd';
  name   |           password
---------+-------------------------------
 bob     | p4ssw0rd
(1 row)

db=# show dejector_mask;
                                                dejector_mask
-------------------------------------------------------------------------------------
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAFAAAAAAAAAAAAAAAAAAAAAAEAAAAAAA
(1 row)
```

And use it

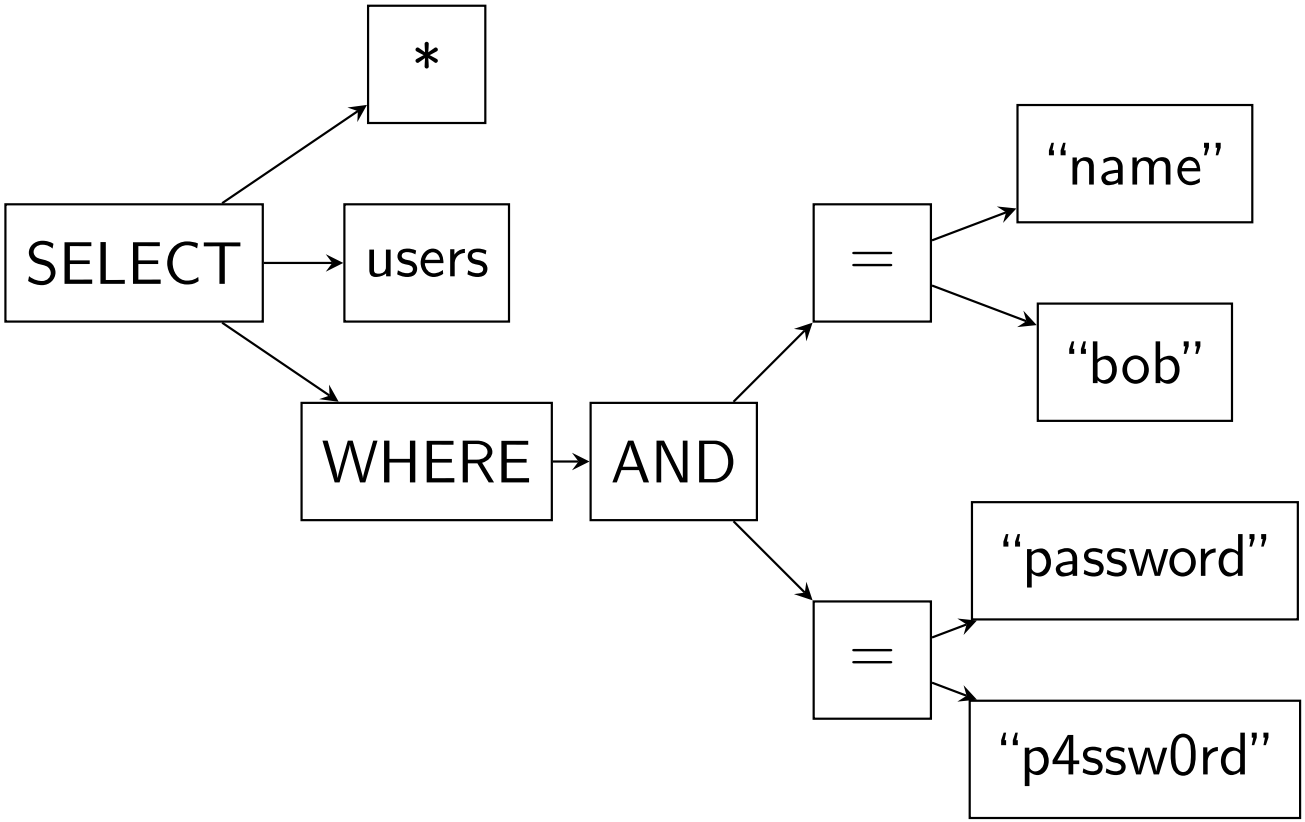
```
db=# set dejector_mask = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAFAAAAA
SET
db=# set dejector_enforcing = 1;
SET
```

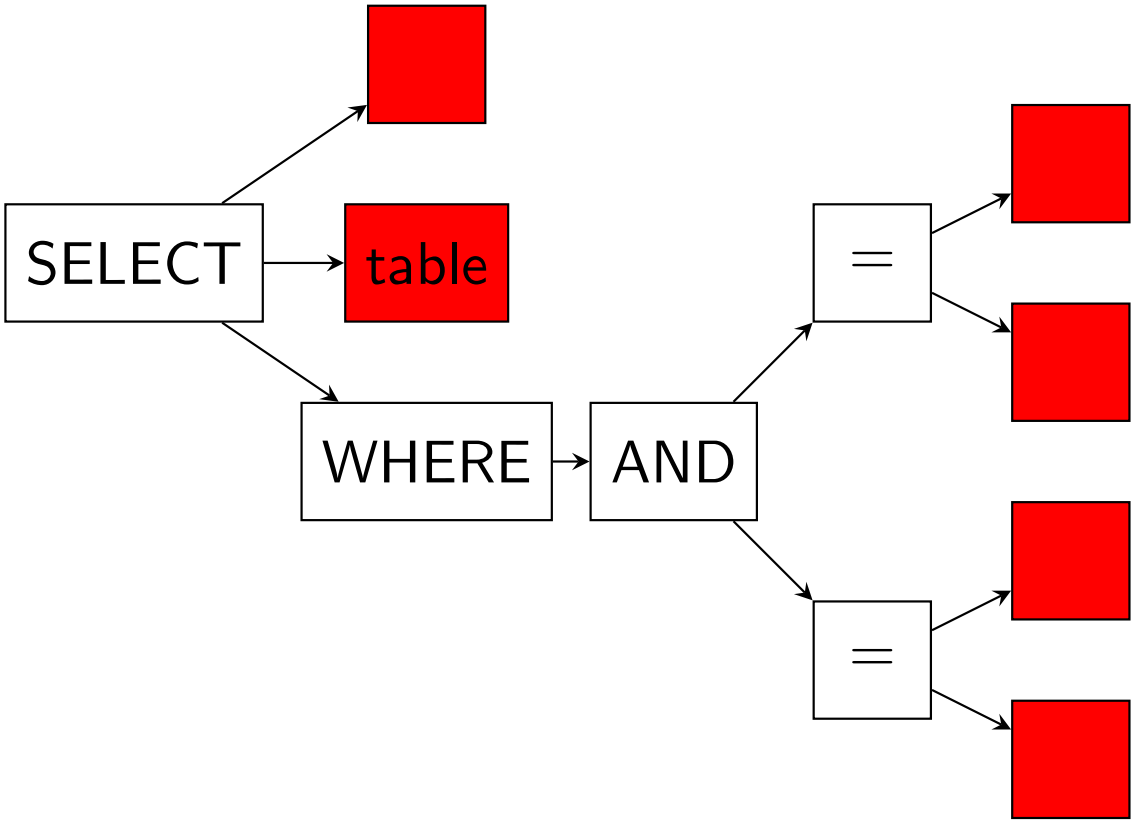
And use it

```
db=# set dejector_mask = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAFAAAAA
SET
db=# set dejector_enforcing = 1;
SET

db=# select * from users where name = 'pastor'
                            and password = 'Spargelzeit!';
  name   |             password
---------+----------------------------------
 pastor | Spargelzeit!
(1 row)
```

And use it

```
db=# set dejector_mask = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAFAAAAA
SET
db=# set dejector_enforcing = 1;
SET

db=# select * from users where name = 'pastor'
                          and password = 'Spargelzeit!';
  name   |             password
---------+----------------------------
 pastor | Spargelzeit!
(1 row)

db=# select * from users where name = 'haxOr' OR 1=1;--'
                          and password = 'p4sswOrd';
ERROR:  SQL injection detected
```


*
SELECT
users
WHERE
AND
=
“name”
“bob”
=
“password”
“p4ssw0rd”


SELECT
table
WHERE
AND
=
=

```
(select tablename
        (WHERE (AND (= (colname string))
                    (= (colname string)))))
```

SHA256[(select *tablename*
(WHERE (AND (= (*colname string*))
(= (*colname string*))))]

Filtering algorithm

- When learning, add the incoming hash to the whitelist
- When filtering, just check the incoming hash against the list

Filtering algorithm

- When learning, add the incoming hash to the whitelist
- When filtering, just check the incoming hash against the list

But that's slow.

Enter bloom filters

- Constant time, constant size, probabilistic datastructure

Enter bloom filters

- Constant time, constant size, probabilistic datastructure
- MAY return false positives, MAY NOT return false negatives

Enter bloom filters

- Constant time, constant size, probabilistic datastructure
- MAY return false positives, MAY NOT return false negatives
- Can store a large number of items for its size

- Instead of the list, a bloom filter

- Instead of the list, a bloom filter
- Split the hash, use as indices

- Instead of the list, a bloom filter
- Split the hash, use as indices
- Voilà! Constant time insertion and lookup

If you have a parse tree...

```
static bool dejector_filter_statement_walker(
                    Node* node, walker_ctx* ctx) {
    if (node == NULL) {
        return false;
    }

    trace('{', node->type);
    raw_expression_tree_walker(
        node, dejector_filter_statement_walker, ctx);
    trace('}', node->type);
    return false;
}
```

If you have a parse tree...

```
static bool dejector_filter_statement_walker(
                    Node* node, walker_ctx* ctx) {
    if (node == NULL) {
        return false;
    }

    trace('{', node->type);
    raw_expression_tree_walker(
        node, dejector_filter_statement_walker, ctx);
    trace('}', node->type);
    return false;
}
```

(and 357 lines of support code)

If you have a parse tree...

```
static bool dejector_filter_statement_walker(
                    Node* node, walker_ctx* ctx) {
    if (node == NULL) {
        return false;
    }

    trace('{', node->type);
    raw_expression_tree_walker(
        node, dejector_filter_statement_walker, ctx);
    trace('}', node->type);
    return false;
}
```

(and 357 lines of support code, 160 of which were base64)

If you don't

If you don't

- Here's a nickel – get yourself a real database.
- Actually, not that hard
- You're really just emitting markers to a stream
- Where the stream is better known as "SHA256"

Come at me, bro!

Beware of bugs in the above code; I have only proved it correct, not tested it

–Donald Knuth

Making a hash of everything

- The current implementation isn't complete

Making a hash of everything

- The current implementation isn't complete
- I should probably be hashing table and column names

Making a hash of everything

- The current implementation isn't complete
- I should probably be hashing table and column names
- And operators

Making a hash of everything

- The current implementation isn't complete
- I should probably be hashing table and column names
- And operators
- What else?

Questions?