

(in)Secure code, Exploits and mitigation AngularJS and MongoDB



Israel Chorzevski
Tech leader, AppSec Labs

Agenda

- ▣ Mongo query manipulation (or SQL Injection)
- ▣ Angular sanitation (or HTML injection)
- ▣ Angular expression injection
- ▣ **Live simulations and exploits for everything...**



Who am I



Israel Chorzevski, over 8 years in the security research

Tech leader at AppSec Labs, Application security experts



AppSec Labs services

- Training (secure coding, hacking)
- Consulting (SDLC)
- Penetration testing
- Mobile (Android, iOS, Windows)

AppSec Labs Learning Management System



HP Software Security Training Center

Online courses and exercises for secure coding



[Login](#)

- [Home](#)
- [KBase](#)
- [Software Security Q&A](#)
- [Courses](#)
- [Tour](#)

Welcome to the **Software Security Training Center!**

The **Software Security Training Center (SSTC)** was created with the intent to enhance your awareness in application security and secure coding.

By understanding the threat landscape and implementing the appropriate countermeasures we are improving the integrity of our products.

Awareness is the name of the game!

SSTC is now available for you to enhance your knowledge of software security and improve our products' security.

Check out your personal training program and download course materials and lab environment to practice what you've learned under Courses in the menu.

Check out our easy-to-use knowledgebase for application security issues you are confronting in real time.

We wish you good luck in revealing the fascinating world of software security

[Click here to take a tour and explore the system's features](#)

HP SW IT Management Security & Trust Office

Contact

Questions, comments, or changes? Tell us: ITM.Securitytrust@hp.com



AppSec Labs

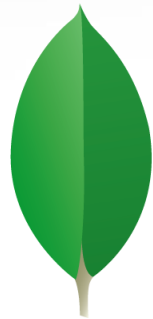
The leading Application Security Company



- 📍 A bunch of Application Security Experts
 - 📍 Ninja Pentesters of Web & Mobile Apps
 - 📍 Elite Trainers for Hacking & Secure coding courses



Mongo DB



mongoDB

No SQL – Mongo DB

- 📄 Document-oriented database
 - 📄 You have no rows, you have a collection of objects (AKA documents)
 - 📄 Each document may have a different structure
 - 📄 Queries are written in BSON (binary JSON)
-
- 📄 Simulation – The structure of the DB

No SQL – Mongo DB

Basic query structure

SQL SELECT Statements

```
SELECT *  
FROM users
```

```
SELECT *  
FROM users  
WHERE status = "A"
```

```
SELECT user_id, status  
FROM users  
WHERE status = "A"
```

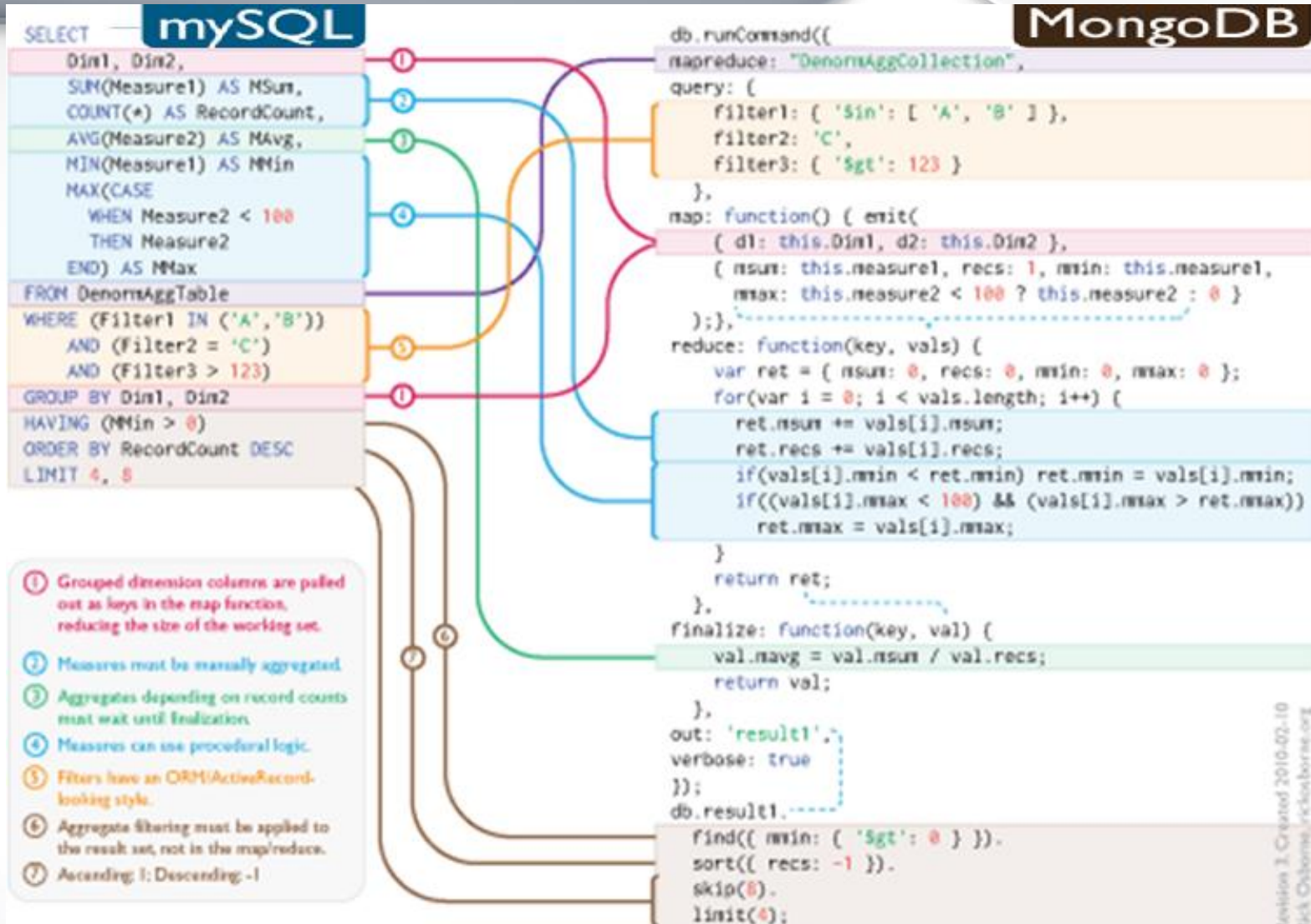
MongoDB find() Statements

```
db.users.find()
```

```
db.users.find(  
    { status: "A" }  
)
```

```
db.users.find(  
    { status: "A" },  
    { user_id: 1, status: 1, _id: 0 }  
)
```


No SQL – Mongo DB



No SQL query string
No SQL Injection?

No SQL – Mongo DB

📄 JSON.parse

📄 Simulations

📄 Manipulation

📄 Denial of Service using \$in

📄 Change \$inc to \$set

No SQL – Mongo DB

☞ And what happens if you have no `JSON.parse`?

☞ Server code:

```
app.all('/login', function(req, res){
  users.find( { user: req.body.u, pass: req.body.p },
    function(err, user){
      console.log ("successfully login");
    });
});
```

☞ Simulation: `node parameter_pollution.js`

No SQL – Mongo DB

☒ HTTP Parameter Pollution

☒ What returns `req.param("fieldX")` if the URL ends with: `?fieldX=abc&fieldX=def`

`['abc', 'def']`

☒ And what will it return if the URL ends with: `?fieldX[abc]=def`

`{abc: def}`

📄 Bypass login simulation

📄 Original query:

```
users.find({user:req.body.u, pass:req.body.p},
```

📄 Malicious bypass

```
<input name="p[$ne]" value="anything" />
```

📄 Final query:

```
users.find({user:req.body.u, pass:{$ne: 1337}},
```

No SQL – Mongo DB

☐ Mitigations:

☐ Use libraries that support parameterized queries

☐ Perform strong input validation using:

☐ Type validation (string is a string and not an object)

☐ Regular expression (white list approach)

Angular



by Google

ANGULARJS



Angular

Data Binding

MVC

Routing

Testing

jQuery

Templates

History

Factories



AngularJS is a full-featured
SPA framework

ViewModel

Controllers

Views

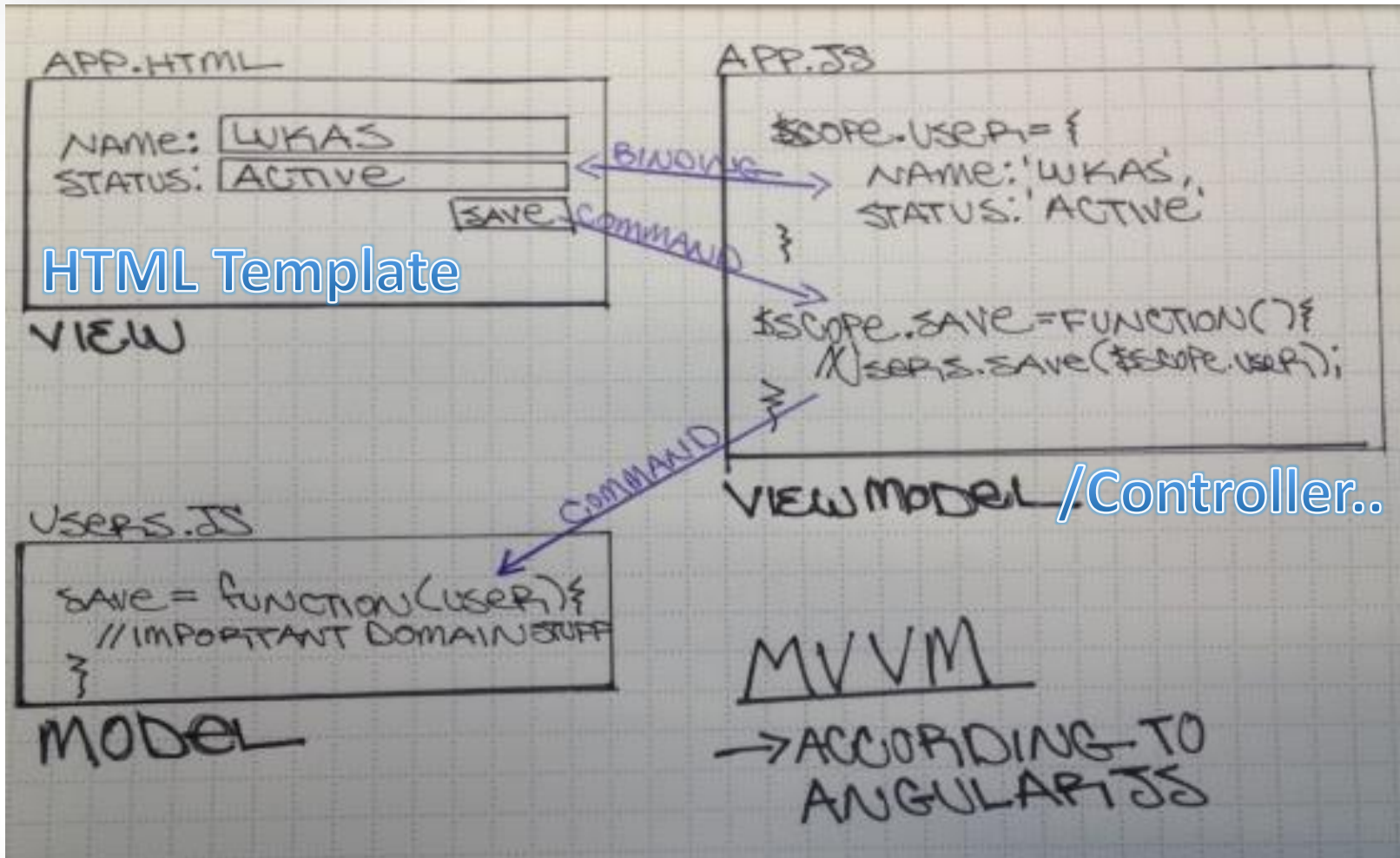
Directives

Services

Dependency Injection

Validation

Angular



Controller, Expression and Binding

▣ ng-controller

- ▣ Set the controller (`$scope`)

▣ ng-bind and `{{expression}}`

- ▣ There are two ways to show a variable content (or function's result) into the page
- ▣ Example: `{{variable}}` ``

▣ ng-model

- ▣ Binds an element to a variable. two-way data binding (`$scope --> view` and `view --> $scope`),
- ▣ Example: `<input type="text" ng-model="variable" />`

▣ Simulation

http://victim-site.com:2000/angular/my_details/

▣ ng-bind and {{expression}}

- ▣ Two ways to show a variable content (or function's result) into the page
- ▣ Example: `{{variable}} `

▣ ng-model

- ▣ Binds an element to a variable. two-way data binding (\$scope --> view and view --> \$scope),
- ▣ Example: `<input type="text" ng-model="variable" />`

Angular Filters

- ☐ ng-bind – Encode the HTML
- ☐ ng-bind-html – Allow filtered HTML
- ☐ linky – Encode the HTML, change links to be clickable

http://victim-site.com:2000/angular/xss_filters2/

- ☐ Direct content updating is still vulnerable !

☐ ng-bind-html – Allow filtered HTML

☐ ng-bind-html is also not safe


☐ Text injection

☐ Paint big pictures

☐ Track users using external pictures

☐ Referrer leakage

Angular – New Injections

 Inject models:

```
{{scared_message | filter}}
```

 Inject scripts:

```
{{myAlert()}}
```

http://victim-site.com:2000/angular/xss/new_vectors2.php

Conclusions

☐ New technologies

- ☐ Introduce new vulnerabilities

- ☐ May reintroduce old vulnerabilities

☐ Strong input validation can assist and is always recommended

☐ Learn, learn, learn...

QUESTIONS ?

Israel@AppSec-Labs.com

THANK YOU !

Israel@AppSec-Labs.com