**Preventing NoSQL Injection Vulnerabilities**

**Introduction**

NoSQL injection vulnerabilities arise when user input is passed into a NoSQL query without being properly sanitized first. Let's walk through the four examples we covered in the last 'chapter' and explain what went wrong and how to fix them.

**String Casting with Input Validation**

**MangoMail**

In the case of MangoMail, this is what the vulnerable code looked like on the server side:

Code: php

...

if ($\_SERVER['REQUEST\_METHOD'] === "POST"):

if (!isset($\_POST['email'])) die("Missing `email` parameter");

if (!isset($\_POST['password'])) die("Missing `password` parameter");

if (empty($\_POST['email'])) die("`email` can not be empty");

if (empty($\_POST['password'])) die("`password` can not be empty");

$manager = new MongoDB\Driver\Manager("mongodb://127.0.0.1:27017");

$query = new MongoDB\Driver\Query(array("email" => $\_POST['email'], "password" => $\_POST['password']));

$cursor = $manager->executeQuery('mangomail.users', $query);

if (count($cursor->toArray()) > 0) {

...

We can see that the problem is $\_POST['email'] and $\_POST['password'] are passed directly into the query array without sanitization, leading to a NoSQL injection which we were able to exploit to bypass authnetication.

MongoDB is [strongly-typed](https://www.techtarget.com/whatis/definition/strongly-typed), meaning if you pass a string, MongoDB will interpret it as a string (1 != "1"). This is unlike PHP (7.4), which is weakly-typed and will evaluate 1 == 1 as true. Since both email and password are expected to be string values, we can cast the user input to strings to avoid anything arrays being passed.

Code: php

...

$query = new MongoDB\Driver\Query(array("email" => strval($\_POST['email']), "password" => strval($\_POST['password'])));

...

Now queries like email[$ne]=x will be cast to "Array" and the attack will fail.

Preventing NoSQL Injection Vulnerabilities

yovecio@htb[/htb]$ php -a

Interactive mode enabled

php > echo strval(array("op" => "val"));

PHP Notice: Array to string conversion in php shell code on line 1

Array

This by itself prevents the NoSQL injection attack from working; however, it would be a good idea to additionally verify that email matches the correct format (to avoid future vulnerabilities/errors). In PHP you can do that like this:

Code: php

...

if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {

// Invalid email

...

}

// Valid email

...

**MangoPost**

On the back end, MangoPost looks slightly different, but it is again the same problem and the same solution.

Code: php

...

if ($\_SERVER['REQUEST\_METHOD'] === "POST") {

$json = json\_decode(file\_get\_contents('php://input'));

$manager = new MongoDB\Driver\Manager("mongodb://127.0.0.1:27017");

$query = new MongoDB\Driver\Query(array("trackingNum" => $json->trackingNum));

$cursor = $manager->executeQuery('mangopost.tracking', $query);

$res = $cursor->toArray();

if (count($res) > 0) {

echo "Recipient: " . $res[0]->recipient . "\n";

echo "Address: " . $res[0]->destination . "\n";

echo "Mailed on: " . $res[0]->mailedOn . "\n";

echo "Estimated Delivery: " . $res[0]->eta;

} else {

echo "This tracking number does not exist";

}

die();

}

...

Cast to a string!

Code: php

...

$query = new MongoDB\Driver\Query(array("trackingNum" => strval($json->trackingNum)));

...

Tracking numbers most likely have a format they follow, so in addition to this cast, we should probably verify that. Let's imagine tracking numbers can contain upper/lowercase letters, digits, and curly braces (/^[a-z0-9\{\}]+$/i). We could create a RegEx to match this and verify tracking numbers like this:

Code: php

...

if (!preg\_match('/^[a-z0-9\{\}]+$/i', $trackingNum)) {

// Invalid tracking number

...

}

// Valid tracking number

...

**String Casting without Input Validation**

**MangoSearch**

The problem in MangoSearch is the same - the query parameter $\_GET['q'] is passed without sanitization into the query array, leading to NoSQL injection.

Code: php

...

if (isset($\_GET['q']) && !empty($\_GET['q'])):

$manager = new MongoDB\Driver\Manager("mongodb://127.0.0.1:27017");

$query = new MongoDB\Driver\Query(array("name" => $\_GET['q']));

$cursor = $manager->executeQuery('mangosearch.types', $query);

$res = $cursor->toArray();

foreach ($res as $type) {

...

Just like in MangoMail, the value of name is expected to be a string, so we can cast $\_GET['q'] to a string to prevent the NoSQLi vulnerability.

Code: php

...

$query = new MongoDB\Driver\Query(array("name" => strval($\_GET['q'])));

...

**Query Rewriting**

**MangoOnline**

Unlike the previous three examples, simply casting to a string will not work in the case of MangoOnline since the exploit did not involve any arrays. As a quick reminder, this is what the back-end code looks like:

Code: php

if ($\_SERVER['REQUEST\_METHOD'] === "POST") {

$q = array('$where' => 'this.username === "' . $\_POST['username'] . '" && this.password === "' . md5($\_POST['password']) . '"');

$manager = new MongoDB\Driver\Manager("mongodb://127.0.0.1:27017");

$query = new MongoDB\Driver\Query($q);

$cursor = $manager->executeQuery('mangoonline.users', $query);

$res = $cursor->toArray();

if (count($res) > 0) {

...

The best option, in this case, is to convert the MongoDB query into one that doesn't evaluate JavaScript while not introducing new vulnerabilities. In this case, it is quite simple:

Code: php

if ($\_SERVER['REQUEST\_METHOD'] === "POST") {

$manager = new MongoDB\Driver\Manager("mongodb://127.0.0.1:27017");

$query = new MongoDB\Driver\Query(array('username' => strval($\_POST['username']), 'password' => md5($\_POST['password'])));

...

According to the [developers](https://www.mongodb.com/docs/manual/reference/operator/query/where/) of MongoDB, you should only use $where if it is impossible to express a query any other way.

If you don't use any queries which evaluate JavaScript in your project, then a good idea would be to completely [disable](https://www.mongodb.com/docs/manual/core/server-side-javascript/#std-label-disable-server-side-js) server-side JavaScript evaluation, which is enabled by default.

**Conclusion**

These steps patched the four vulnerable websites against NoSQL injections. Traditional SQL databases have parameterized queries which are an excellent way to prevent injection but preventing MongoDB / NoSQL injection is not as simple. The best you can do as a developer is:

1. Never use raw user input. Always sanitize, for example, with a white list of acceptable values.
2. Avoid using JavaScript expressions as much as possible. Most queries can be written with regular query operators.