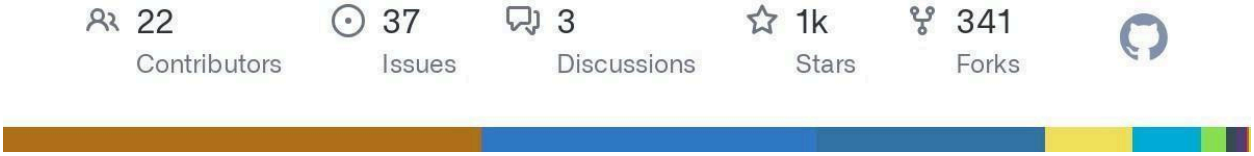
crAPI

Security Assessment Findings Report

OWASP/**crAPI**

completely ridiculous API (crAPI)



Date: OCT 24th to 31th , 2024 **By Mohammed Khalid**

Table of Contents

Confidentiality Statement

This document is a confidential property of **crAPI**.This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent.

Disclaimer

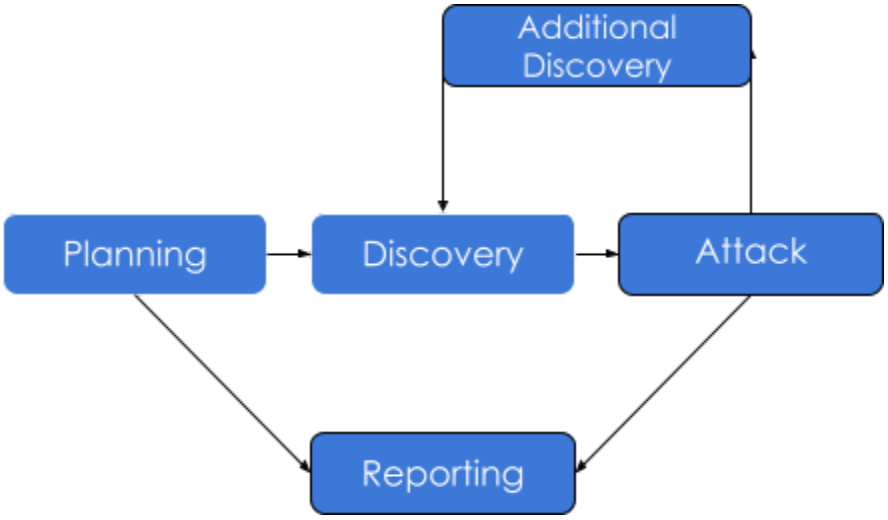
This report is the result of a comprehensive security assessment conducted on **OWASP crAP** as of **October 31**. The scope of this assessment was limited to the web application and its immediate supporting infrastructure as agreed upon. The assessment was conducted based on the methodologies and practices that are current as of **October 31.** Any changes to the web application or its environment after the date of the assessment are not covered under this report.

Contact Information

Name	Title	Contact Information
Mohammed Khalid	*Pentester*	*mdkhalid90366@gmail.com*

Assessment Overview

From **Oct 24th, 2024 to Oct 31st, 2024**, I engaged **crAPI** to evaluate the security posture of its infrastructure compared to current industry best practices. All testing performed is based on the NIST SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customised testing frameworks.



Phases of penetration testing activities include the following:

- **Planning** – Customer goals are gathered and rules of engagement are obtained.
- **Discovery** – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- **Attack** – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- **Reporting** – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

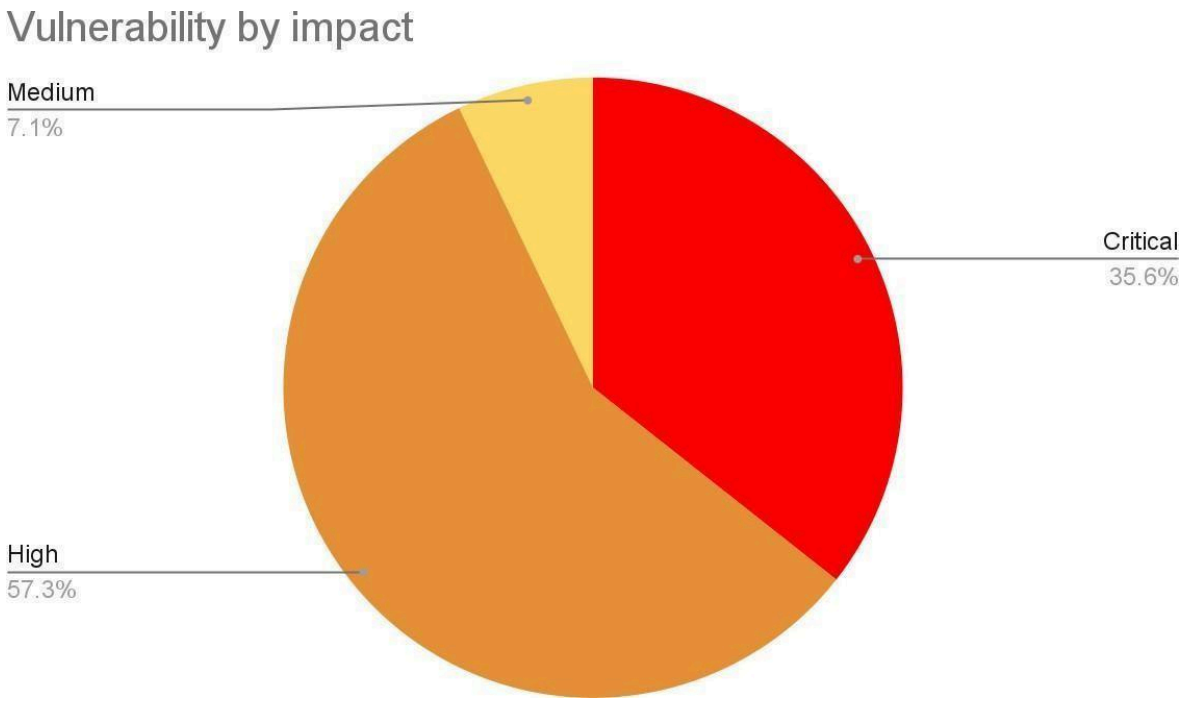
Scope

Scope	Scope type	Start date	End Date
*.localhost:8888	Web	24th OCT	31st OCT
*.localhost:8025	Web	24th OCT	31st OCT

Executive Summary

The Vulnerability Assessment and Penetration Testing (VAPT) conducted on **crAPI** revealed key security insights as of **31 October 2024**. While the assessment identified vulnerabilities within the web application and its infrastructure, it's essential to note that not all potential weaknesses may have been uncovered. This report provides recommendations to enhance security, but implementation should be approached cautiously. Confidentiality and limitations apply.

Vulnerability by Impact



This chart shows the percentage of vulnerability based on risk level. Please go through it to get an insight on risk.

Penetration Test Findings

Index

SL. NO	Vulnerability	Affected Component	CVSS SCORE	Risk
1	Broken Object Level Authorisation	Location endpoint	7.5	High
2	Broken Object Level Authorisation	Contact Mechanic	7.5	High
3	Broken User Authentication	Reset password of another user	9.1	Critical
4	Excessive Data Exposure	data exposure on /posts/recent	7.5	High
5	Excessive Data Exposure	data exposure on /user/videos	7.5	High

6	Rate Limiting	Contact mechanic	7.5	High
7	Broken Function Level Authorization	video	9.1	Critical
8	Mass Assignment	Order return	7.5	High
9	Mass Assignment	order return quantity	7.5	High
10	Mass Assignment	Video properties	7.5	High
11	Ssrf	Contact mechanic	6.5	Medium
12	NoSQL injection	Coupon	9.1	Critical
13	Unauthenticated access	orders	9.1	Critical
14	Token Tampering	JWT Token	9.1	Critical

Vulnerability no. 1

Name of Vulnerability: *Broken Object Level Authorisation (BOLA)*

- **Affected Component:** *Location Endpoint*

- **Affected URL:**

  http://127.0.0.1:8888//identity/api/v2/vehicle/{{vechicle_id}}/location

Description Of Vulnerability:

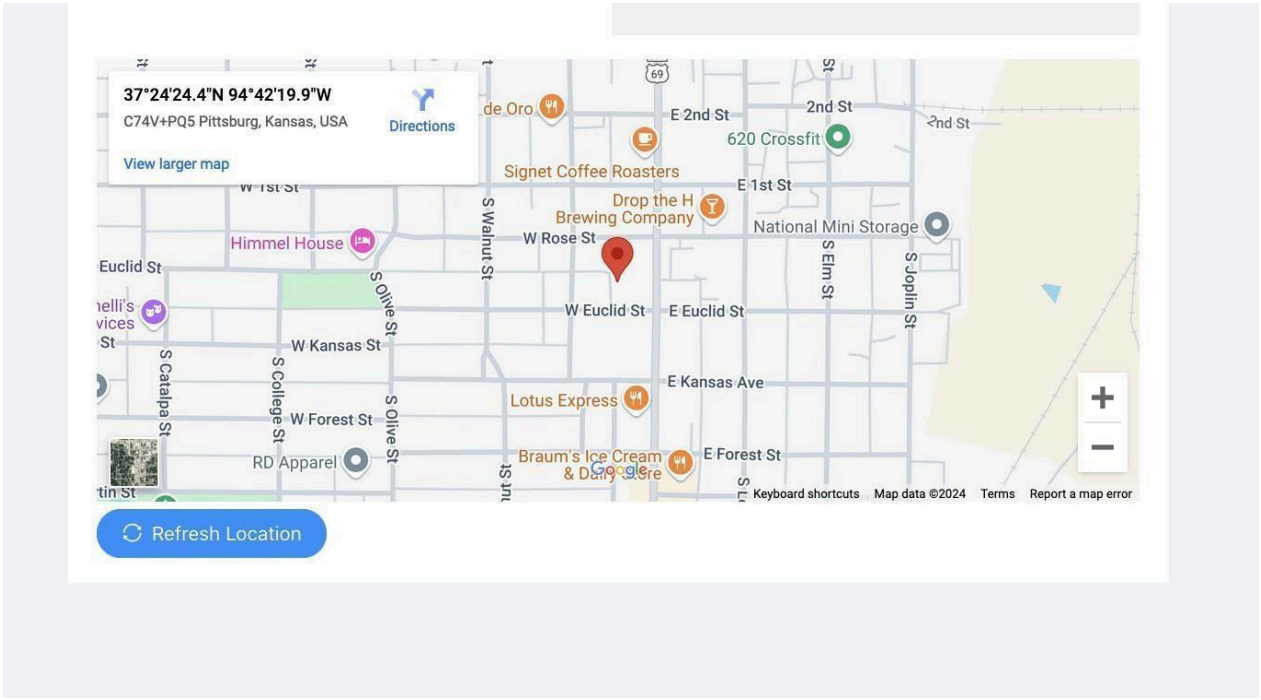
This is a vulnerability caused because of improper authorisation checks by the web app. For a comprehensive approach let me explain the vulnerability(security issue) with a real-life scenario:
Consider you are at an e-commerce website to purchase some products, imagine you clicked on the profile picture to get directed to your profile, on looking at the URL you are seeing something like this: **http://example.com/user/2**
Here the hacker's entry point is the number or id of user given in this URL. A simple way to exploit this is just to change

the ID given. For example, changing it to 3 to get details of another user.
The above given example is specific to web hacking it is called **IDOR**(Insecure direct object reference). The API version of the vulnerability is called **BOLA** (Broken object level authorisation). It's what I found here. The only difference is that it's an API endpoint.
The vulnerability arises because the web app is returning the response without properly checking if the user has access to the resource that he is accessing.

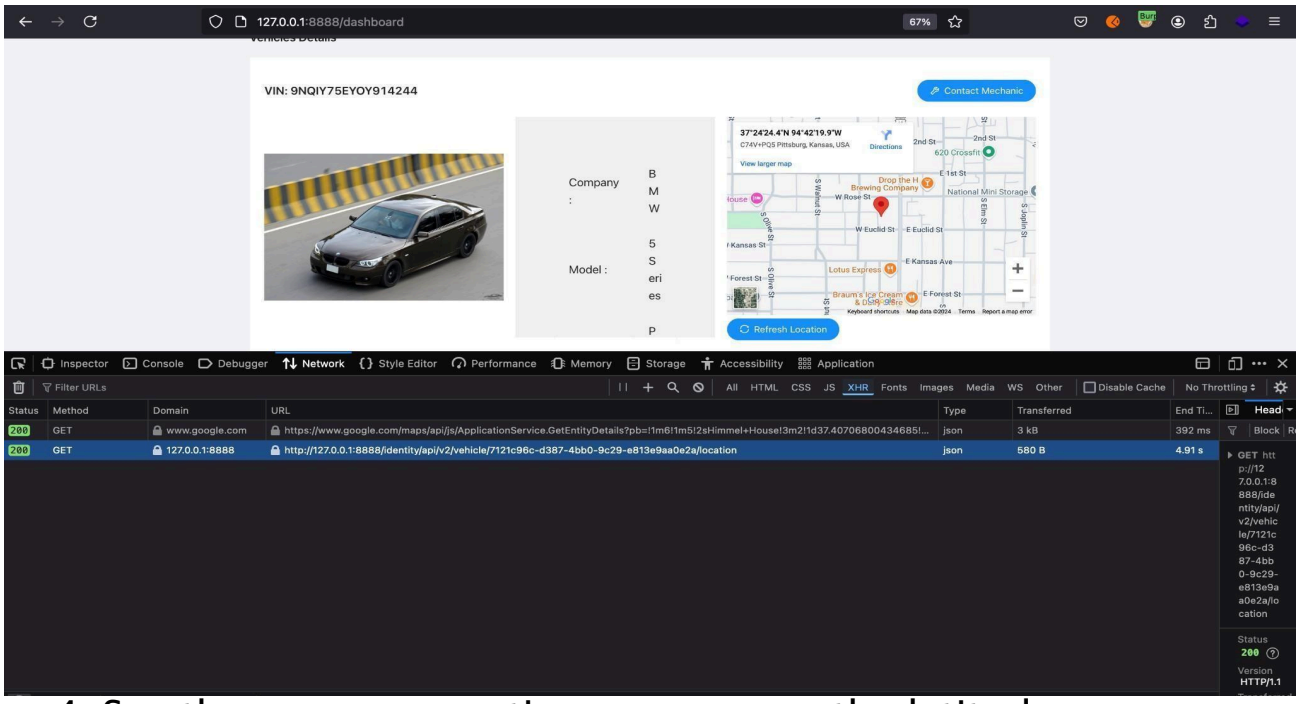
Steps to Reproduce the Vulnerability:

Tools used: Web inspector

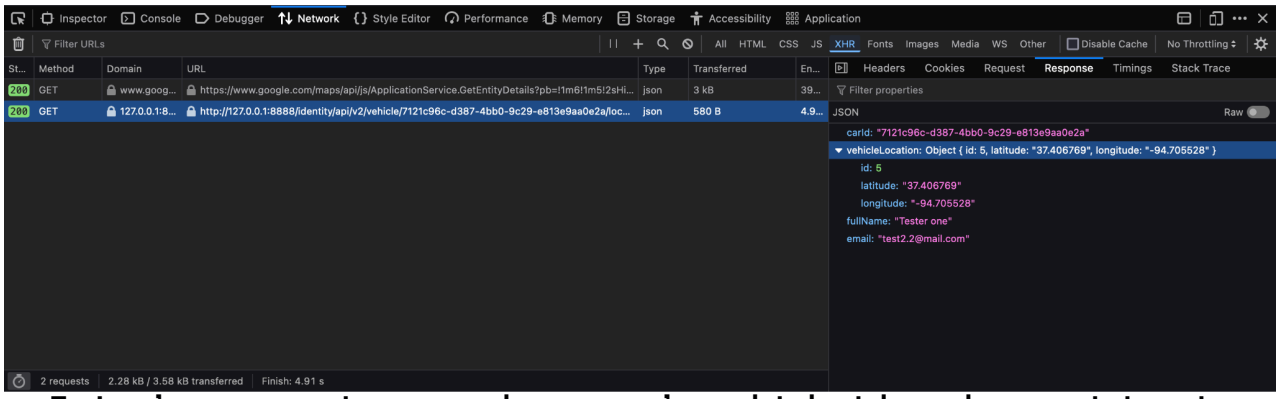
1. Navigate to this URL http://locaalhost:8888/dashboard
2. Now you can see a button bottom left of the page called Refresh Location



3. Right-click on Inspect and go to the network tab

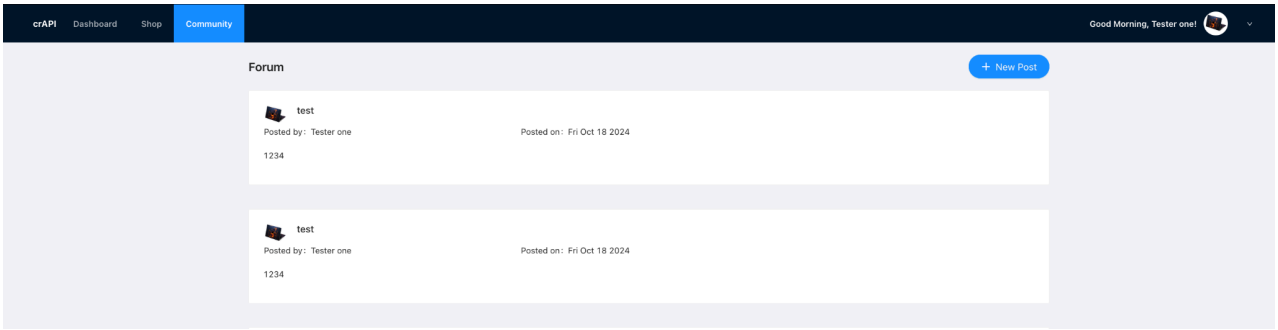


4. See the response section, we can see the latitude, longitude etc of the vehicle, this is got by giving the vehicle ID into the URL:

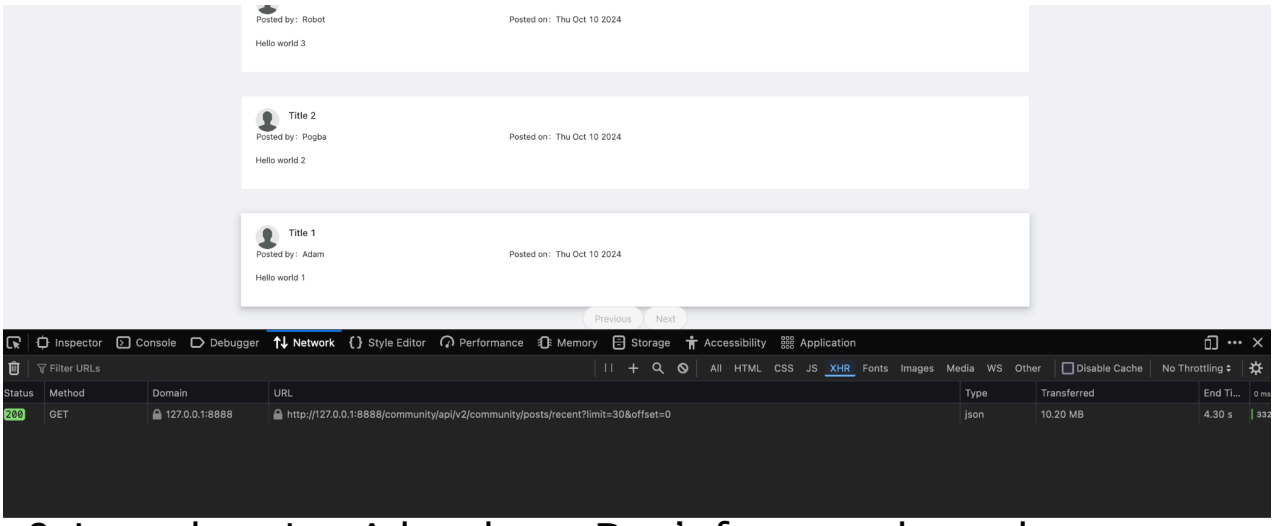


5. Let's try getting another user's vehicle id and see giving it into the URL will return a response that contain these kind of information.For that we need to get another user's vehicle id.

6. For that go to the communities tab

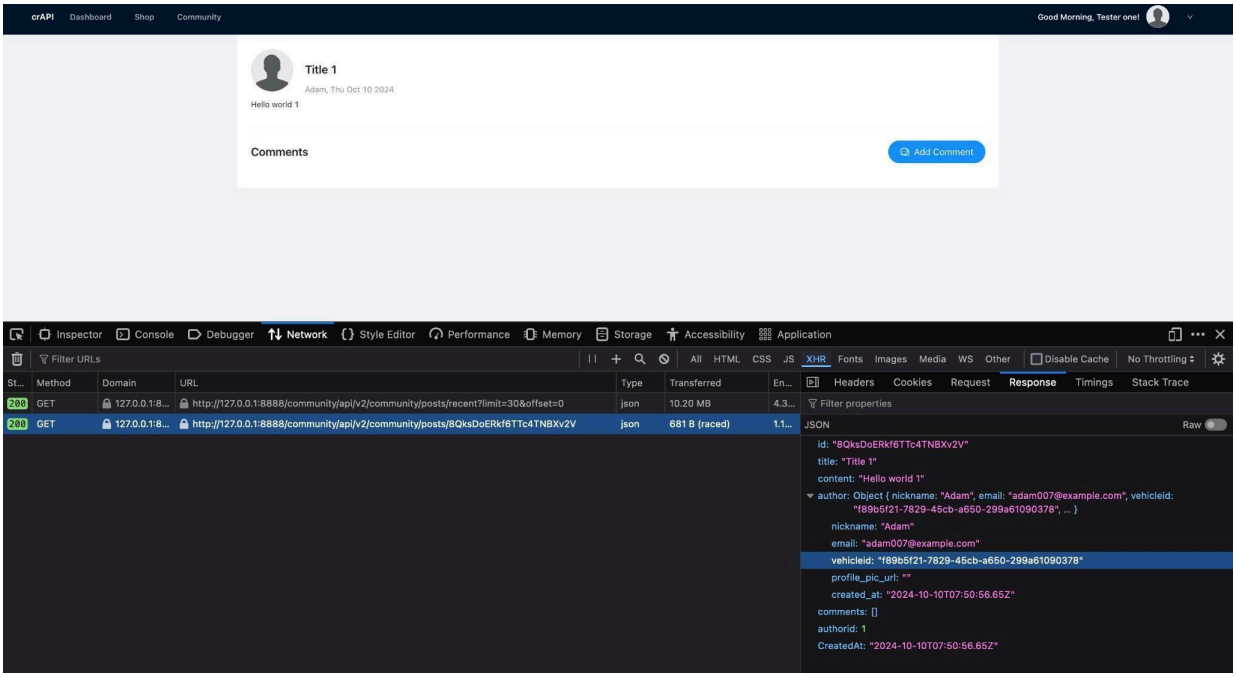


7. Now go to any user's post that's not you



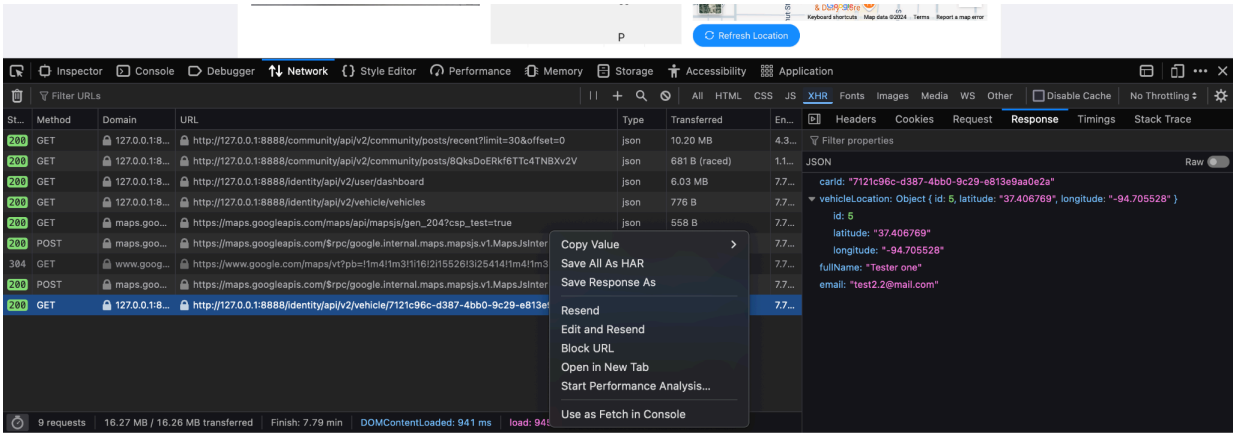
8. I am choosing Adam here, Don't forget to keep the inspect tab open

9. On clicking the post you can see that a request on the network tab popped up, closely look at its response section you can see that vehicle id is there as JSON response.

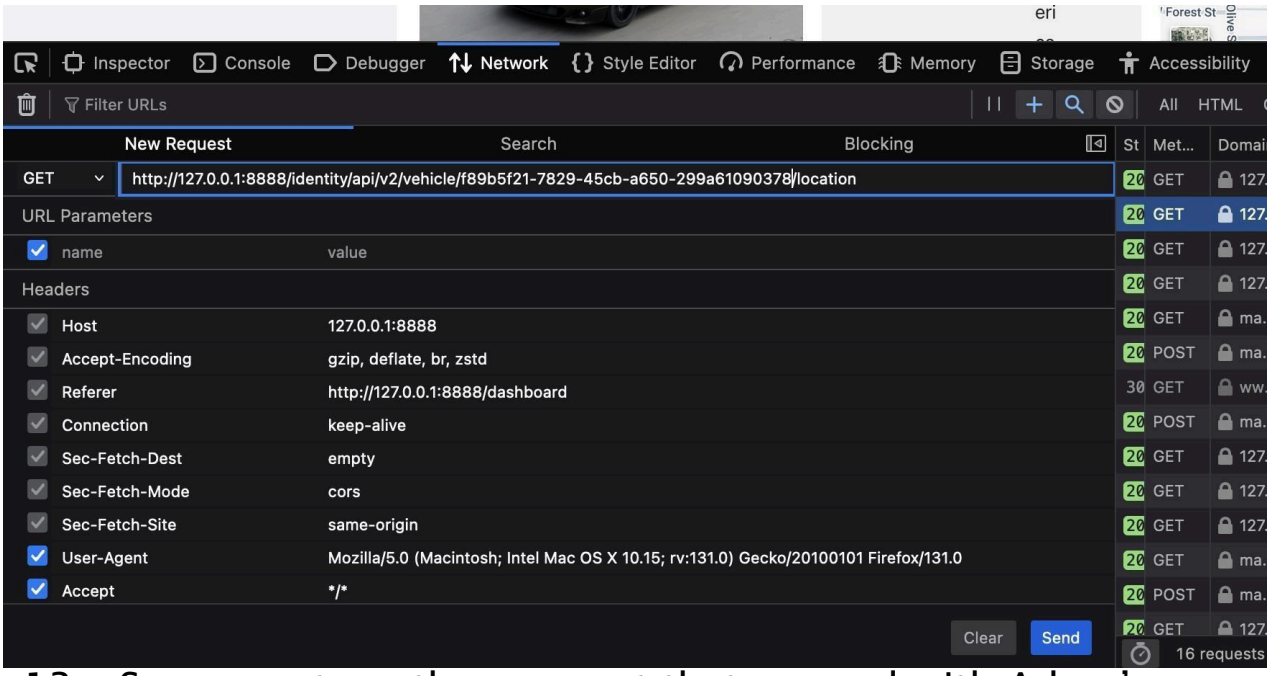


10. Copy the id, now we can go back to where we started please repeat up to step 4

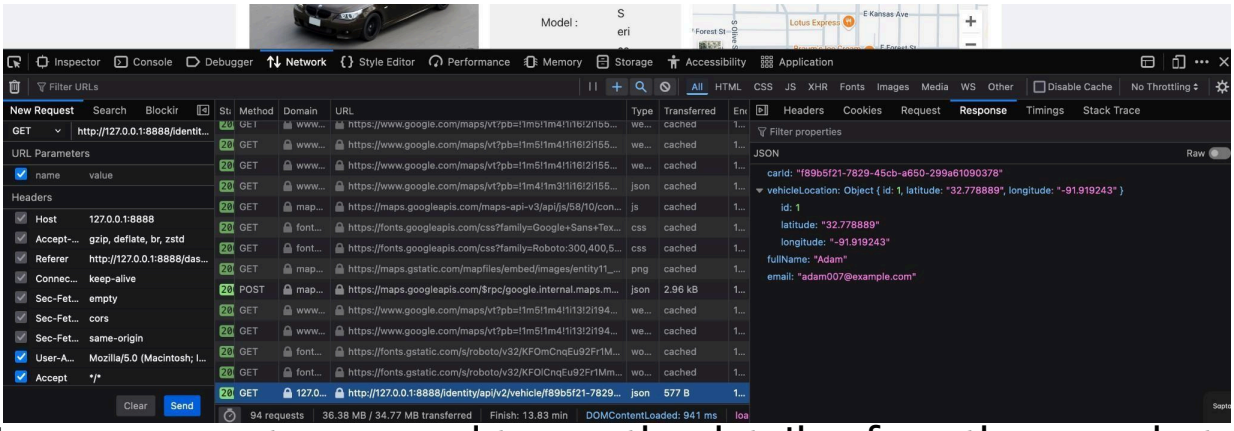
11. Right Click on the request and select edit and resend



12. Here I changed, vehicle id of mine to Adam's and clicked send



13. See we got another request that popped with Adam's vehicle details on it.



Here we are not supposed to see the details of another user but it happened. It's indeed a vulnerability

Impact

Company & Customer:

1. Possible reputational damage
2. This can result in disruption of operation.
3. Sensitive data of customers will be exposed which can cause much more issues.
4. Trust degradation.
5. Chance to stay away from the services or products that the company offers.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- Legal issues consist of violation of GDPR or CCPA as sensitive personal data is exposed.
- The company could face lawsuits, fines, and reputational damage for not securing access controls properly.
- Individuals affected by BOLA could seek legal redress for privacy violations or data breaches.

Steps for mitigation

1. Only users with proper rights should access or modify the data, therefore implement a function to check the user logged in and that the user ID specified here relates to the same, if so then only return data else reject.

2. Enable role-based access control to have granular control over the resources.

3. Validate user permissions on every request

4. Replace the api1_id with non-sequential identifiers eg: here the id goes by 1,2,3 etc it should be changed to using non-sequential identifiers like GUIDs.

5. Avoid exposing ids (vehicle_id) in URL, instead use tokens.

Vulnerability no. 2

of Vulnerability: Broken Object level authorisation (BOLA)

- **Affected Component:** Contact mechanic function
- **Affected URL:**

http://127.0.0.1:8888/workshop/api/mechanic/mechanic_report?report_id={{id}}

Description Of Vulnerability:

This is a vulnerability caused because of improper authorisation checks by the web app. For a comprehensive approach let me explain the vulnerability(security issue) with a real-life **scenario**:
Consider you are at an e-commerce website to purchase some products, imagine you clicked on the profile picture to get directed to your profile, on looking at the URL you are seeing something like this: **http://example.com/user/2**

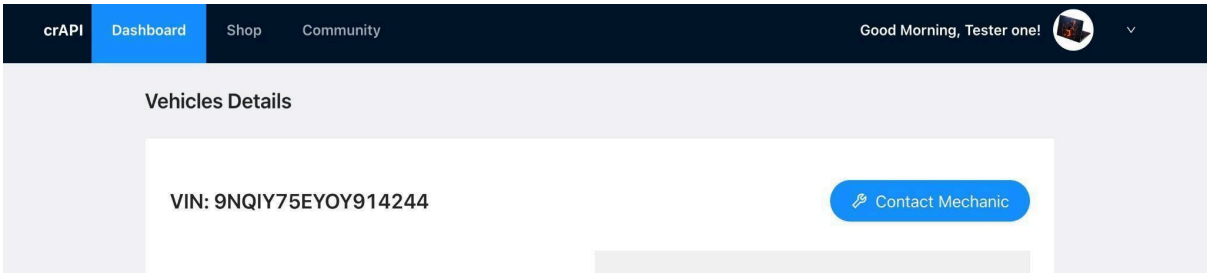
Here the hacker's entry point is the number or id of user given in this URL. A simple way to exploit this is just to change the ID given. For example, changing it to 3 to get details of another user.
The above example is specific to web hacking it is called **IDOR**(Insecure direct object reference). The API version of the vulnerability is called **BOLA** (Broken object level authorisation). It's what I found here. The only difference is that it's an API endpoint.
The vulnerability arises because the web app is returning the response without properly checking if the user has access to the resource that he is accessing.

Steps to Reproduce the vulnerability:

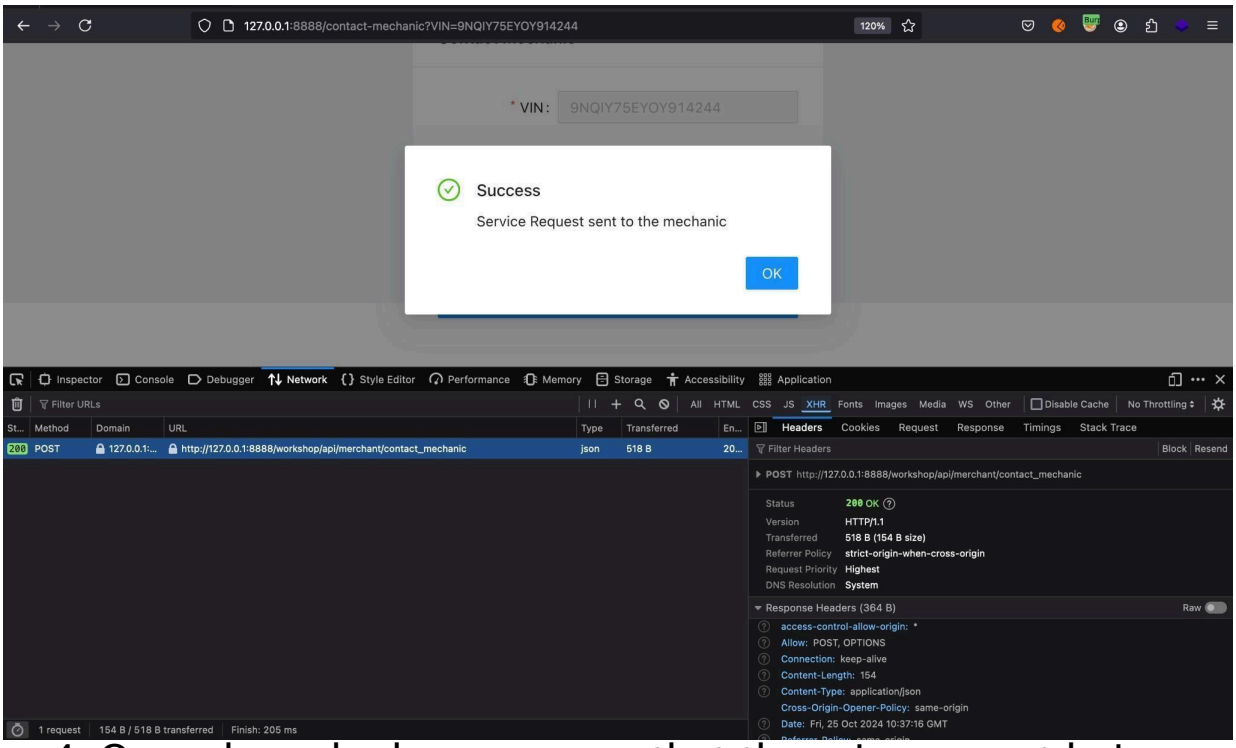
Tools used: Web inspector

1. Go to the dashboard: http://127.0.0.1:8888/dashboard

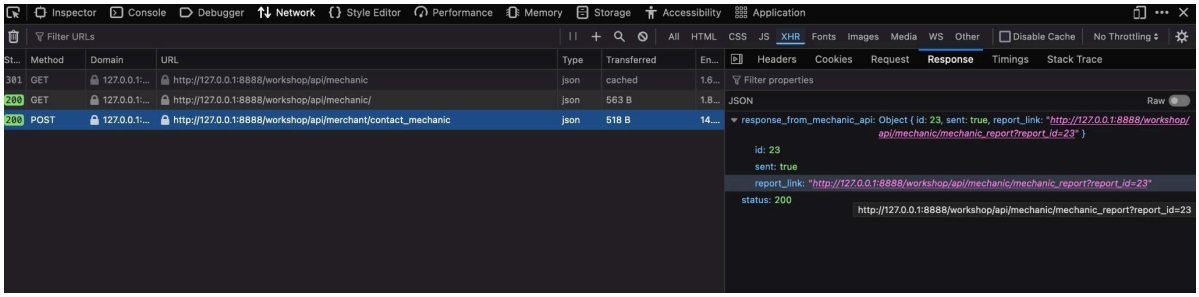
2. Click on the Contact Mechanic button



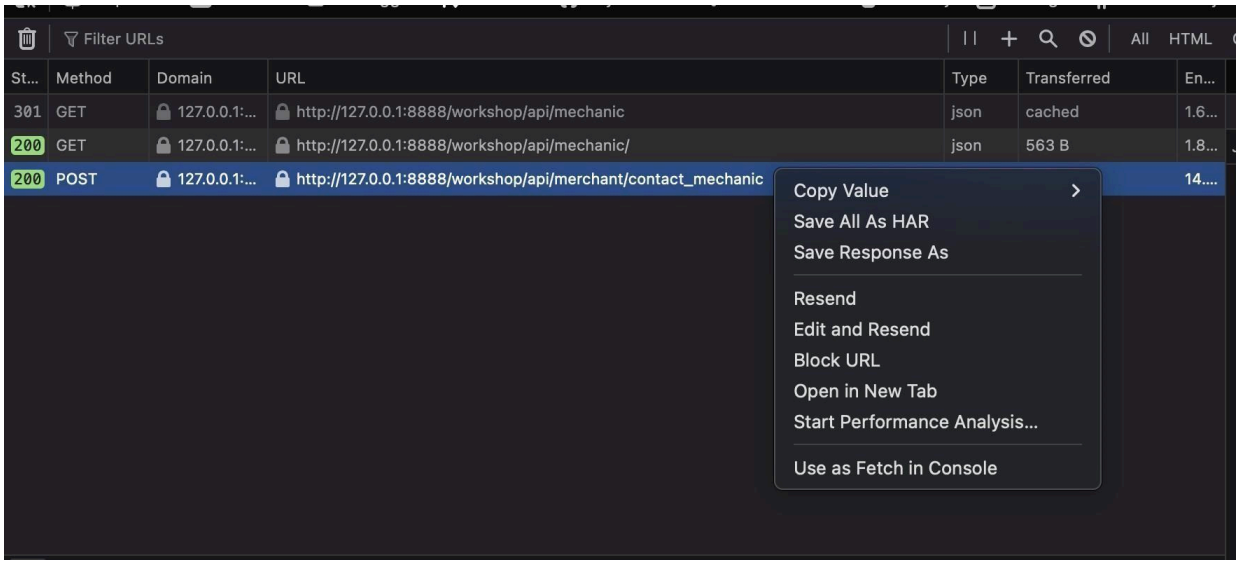
3. Fill in the details before submitting the form don't forget to turn the web inspector on:



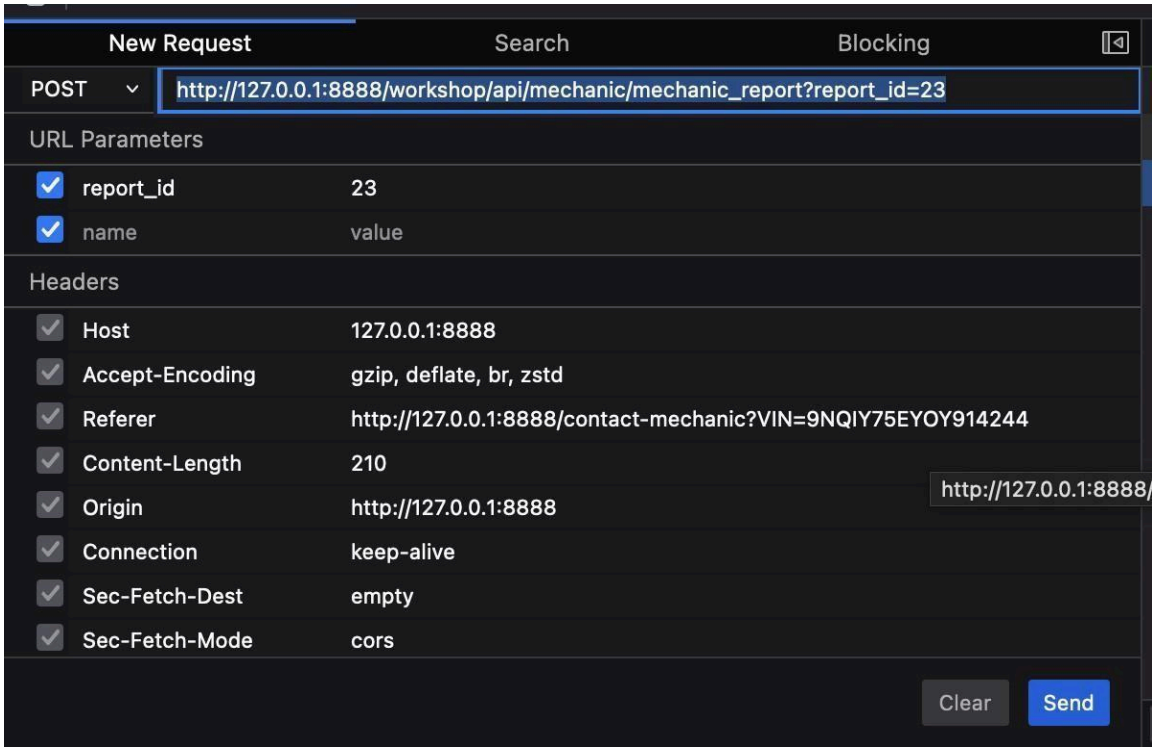
4. On a closer look we can see that there is a report being generated.



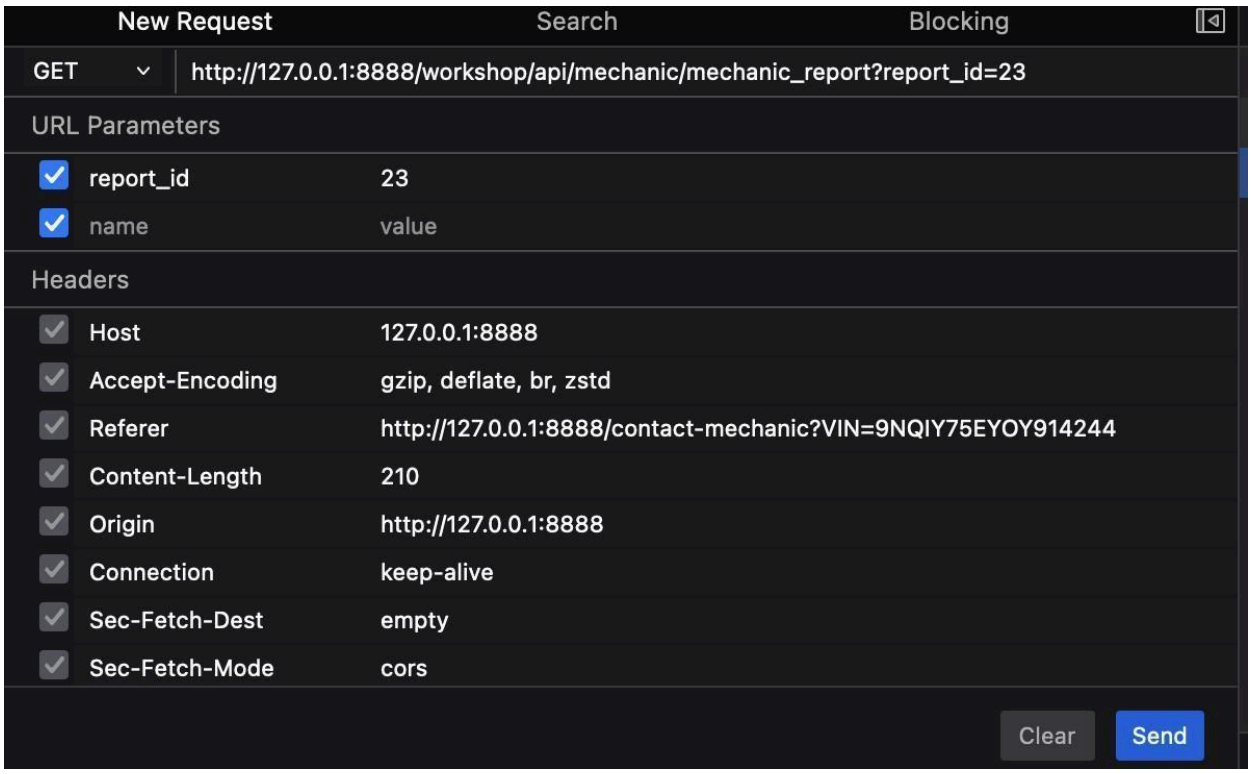
5. We need to see the report but giving it on browser will not work, so let's edit the contact_mechanic request that we just saw, for that right click on the request and edit it.



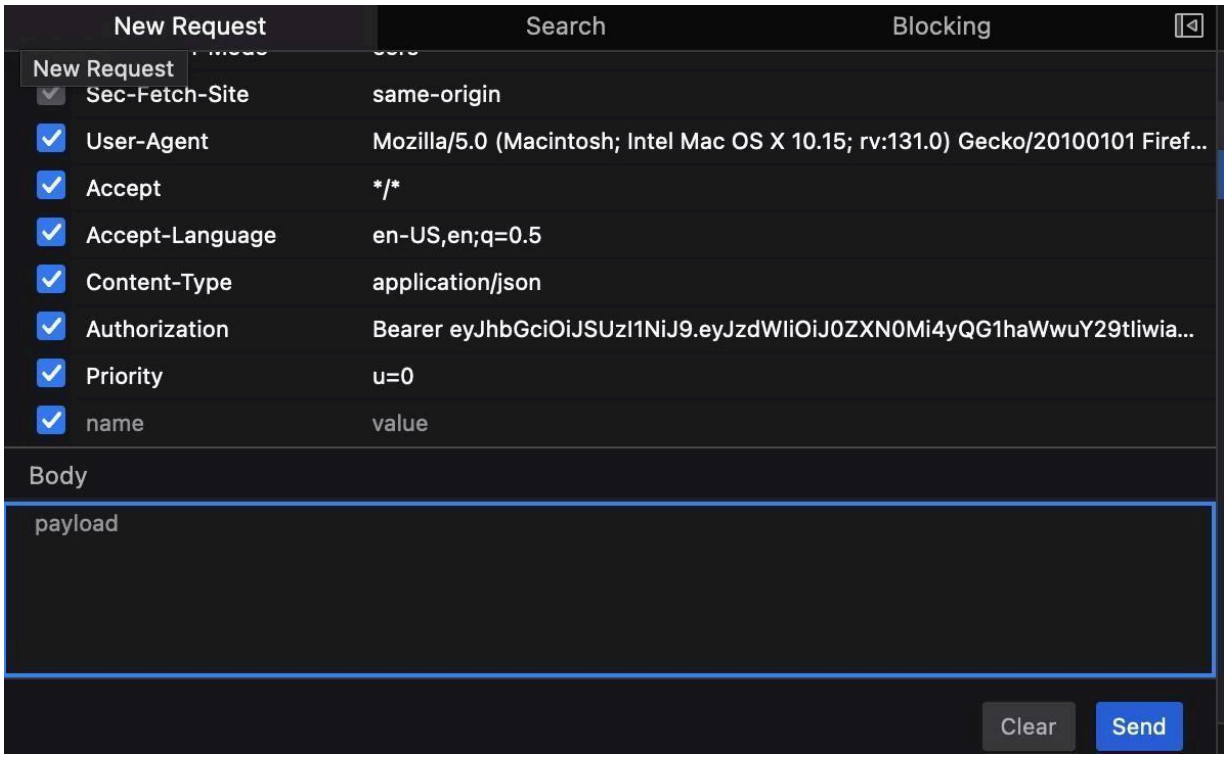
6. Now copy the link and paste it like this:



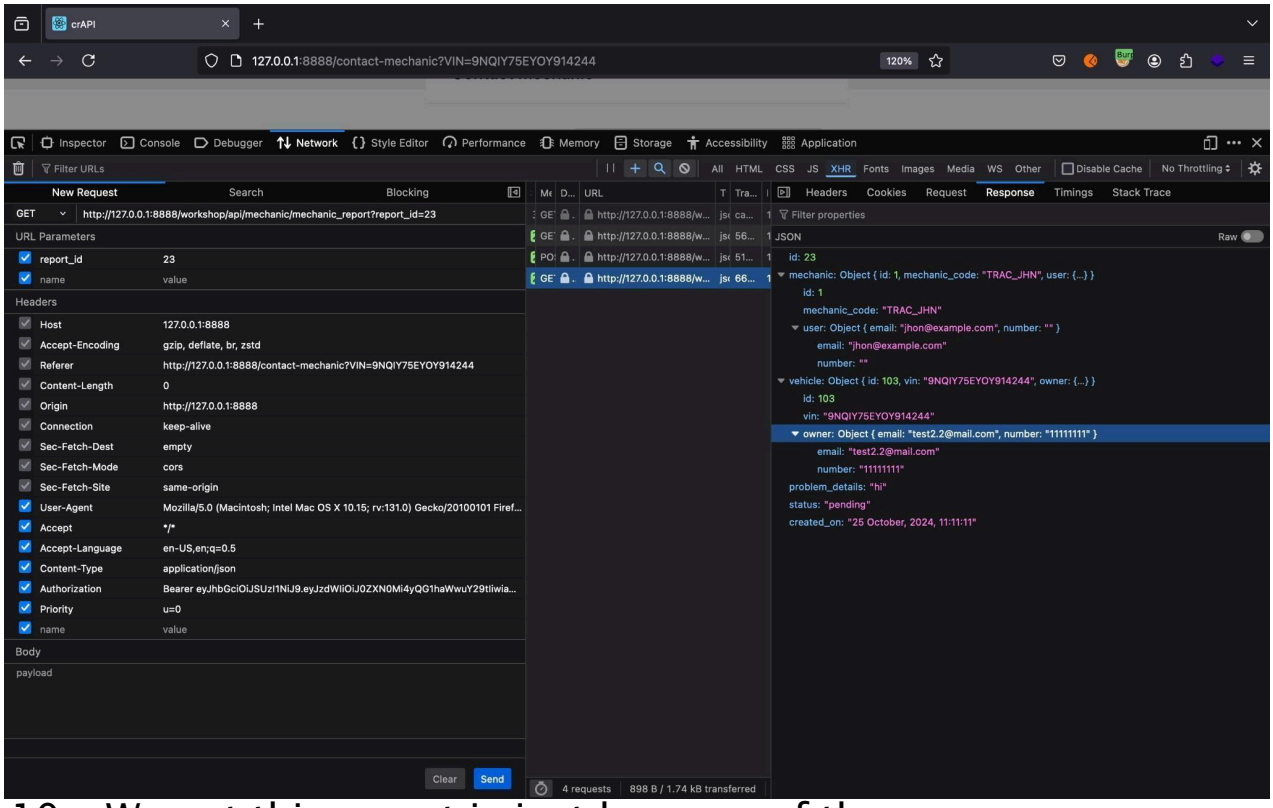
7. Now change the post method (top left corner) to GET



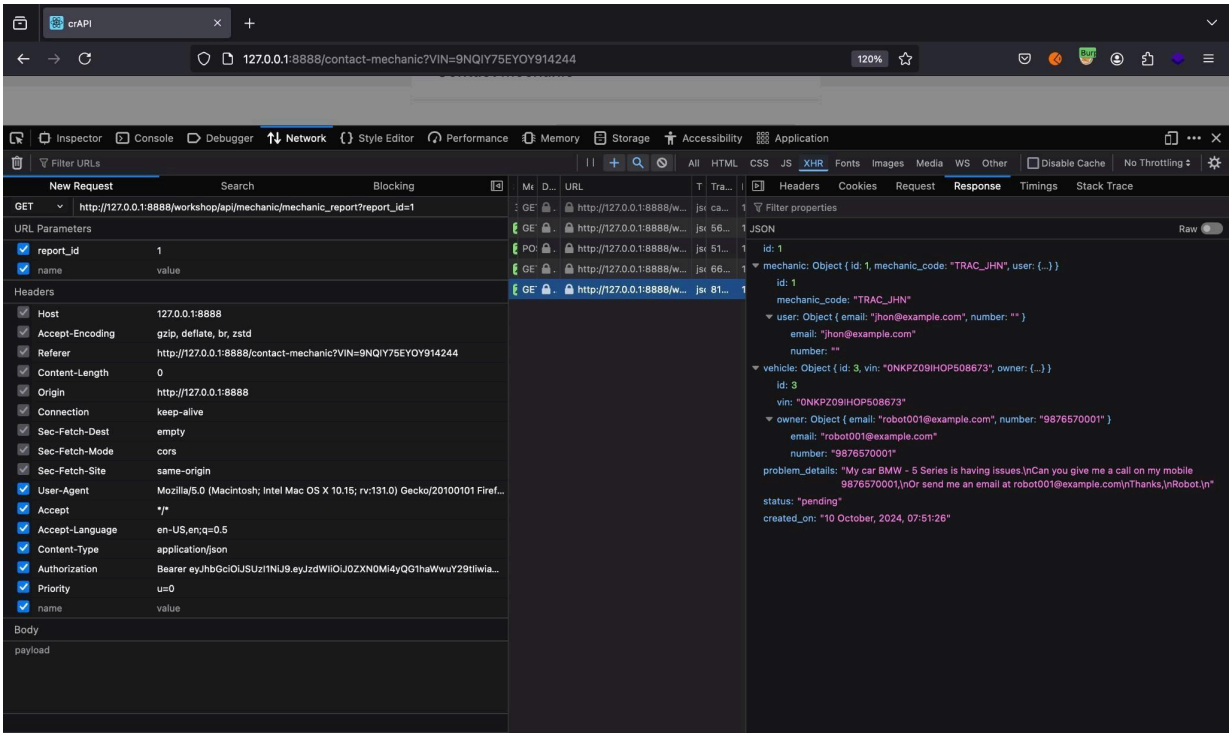
8. Remove body part (payload) and click send



9. This is the response we got



10. We got this report is just because of the id parameter let's try changing that to '1'



This is indeed a vulnerability because by changing the id parameter here we can access reports of any user unauthorised, This is a classic example of **BOLA** vulnerability.

Impact

Company & Customer:

1. Possible reputational damage
2. This can result in disruption of operation.
3. Sensitive data of customers will be exposed which can cause much more issues.
4. Trust degradation.
5. chance to stay away from the services or products that the company offers.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- Legal issues consist of violation of GDPR or CCPA as sensitive personal data is exposed.
- The company could face lawsuits, fines, and reputational damage for not securing access controls properly.
- Individuals affected by BOLA could seek legal redress for privacy violations or data breaches.

Steps for mitigation

1. Only users with proper rights should access or modify the data, therefore implement a function to check the user logged in and that the user ID specified here relates to the same, if so then only return data else reject.

2. Enable role-based access control to have granular control over the resources.

3. Validate user permissions on every request

4. Replace the api1_id with non-sequential identifiers eg: here the id goes by 1,2,3 etc it should be changed to using non-sequential identifiers like GUIDs.

5. Avoid exposing ids (report_id) in URL, instead use tokens.

Vulnerability no. 3

Name of Vulnerability: **Broken User Authentication**

- **Affected Component:** Reset the password of a different user
- **Affected URL:**

Description Of Vulnerability:

This is a vulnerability happens where there is a flaw in the authentication mechanism, so basically authentication is the process of verifying the identity of the user is indeed what they say.

Here in this specific case, a user can reset the password of another user just by getting their email address, which can be obtained from just peeking around.

I will provide a detailed step on how to reproduce the vulnerability.

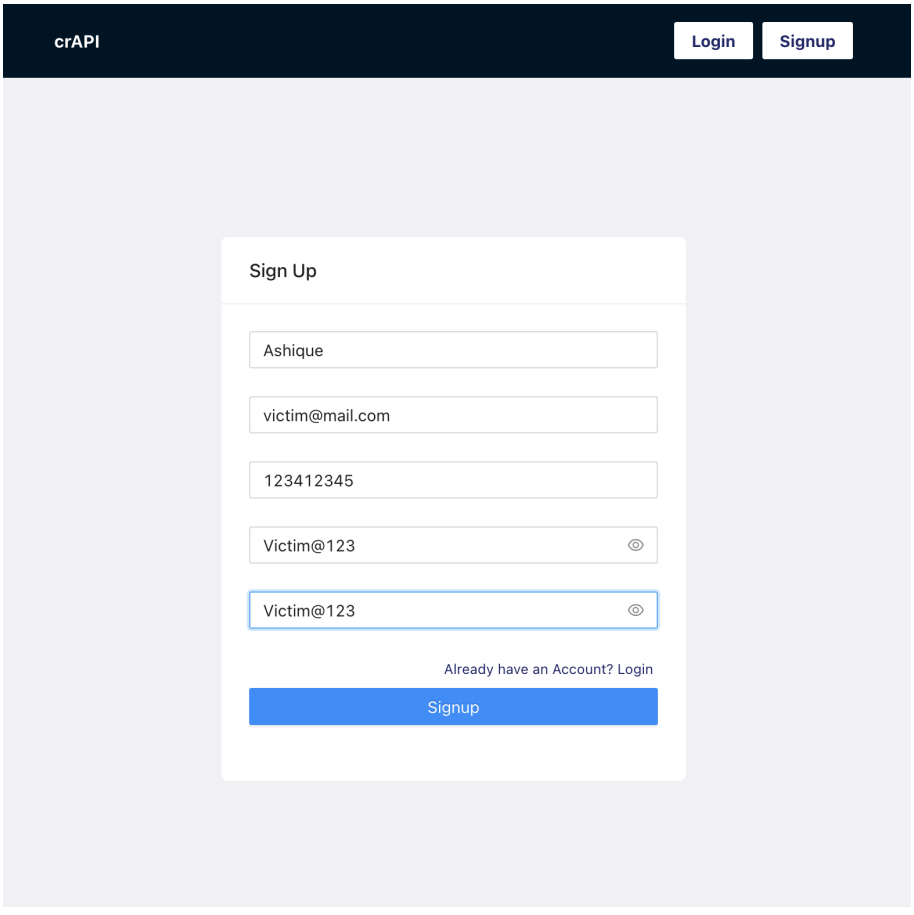
Steps to Reproduce the vulnerability:

Tools used(Expecting basic knowledge of these tools):

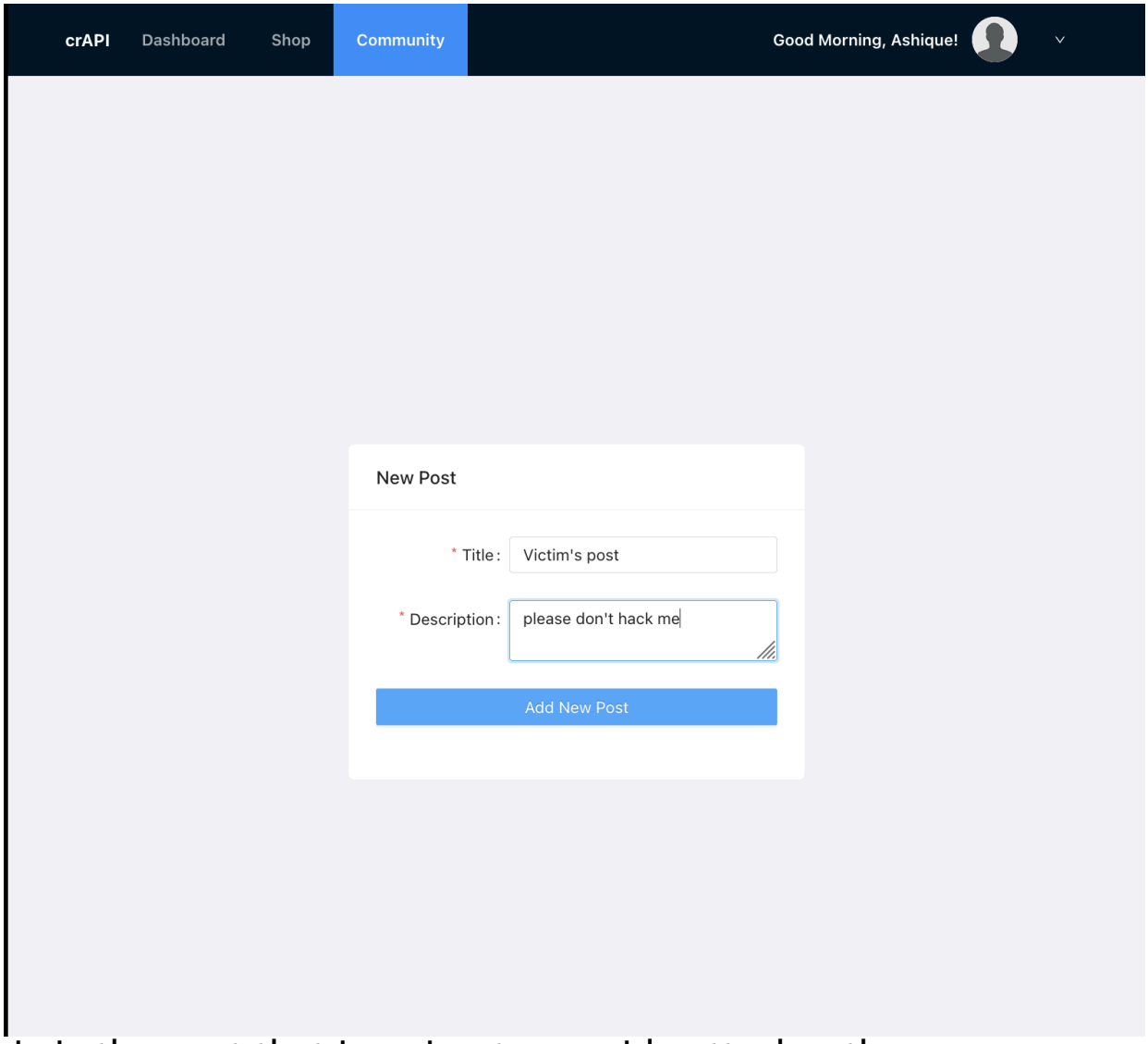
- Web inspector
- Burp suite
- ffuf

Steps:

1. Logout if already authenticated
2. Now go to the login page and create a victim user

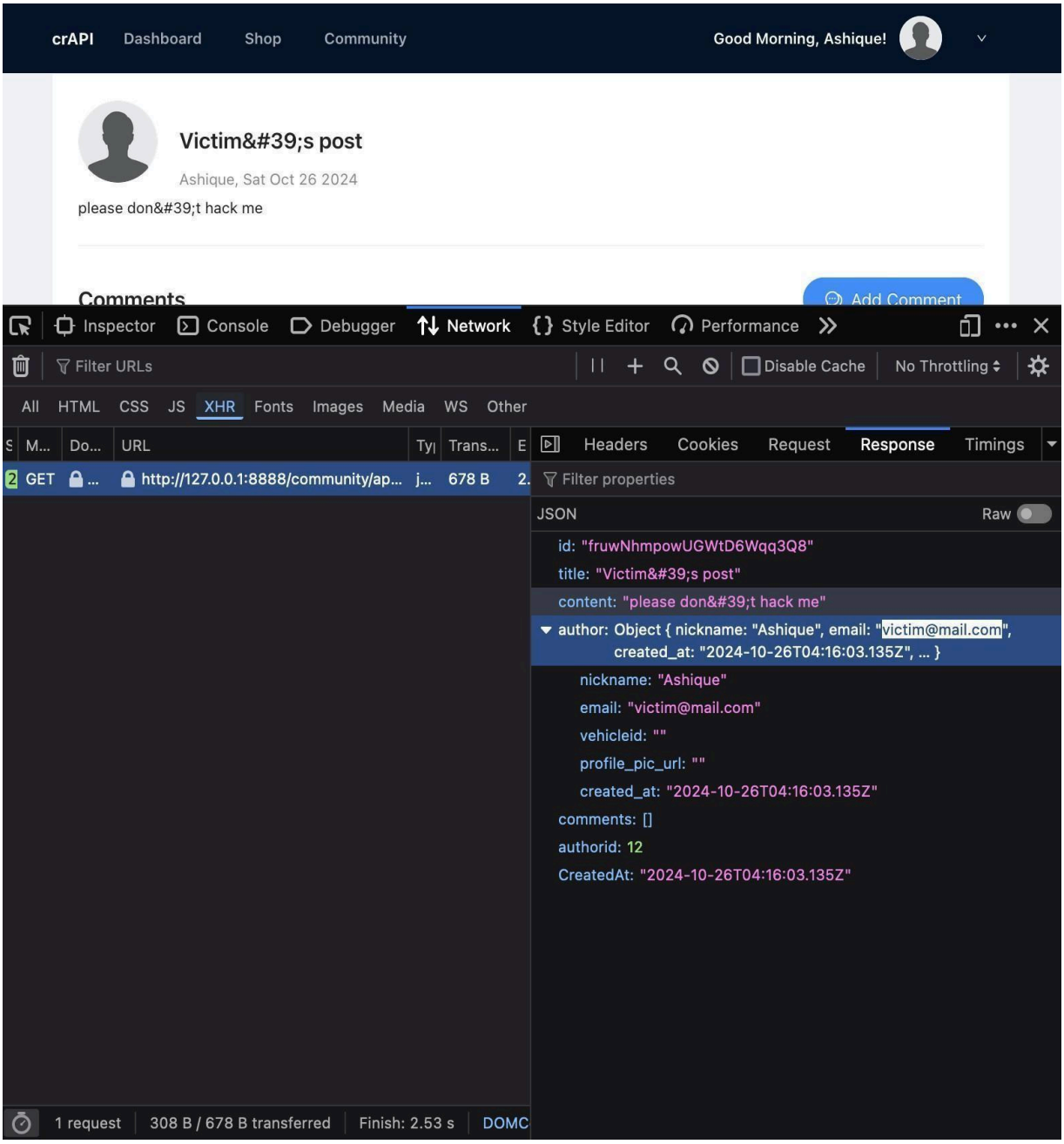


3. Now add a post on the community section.



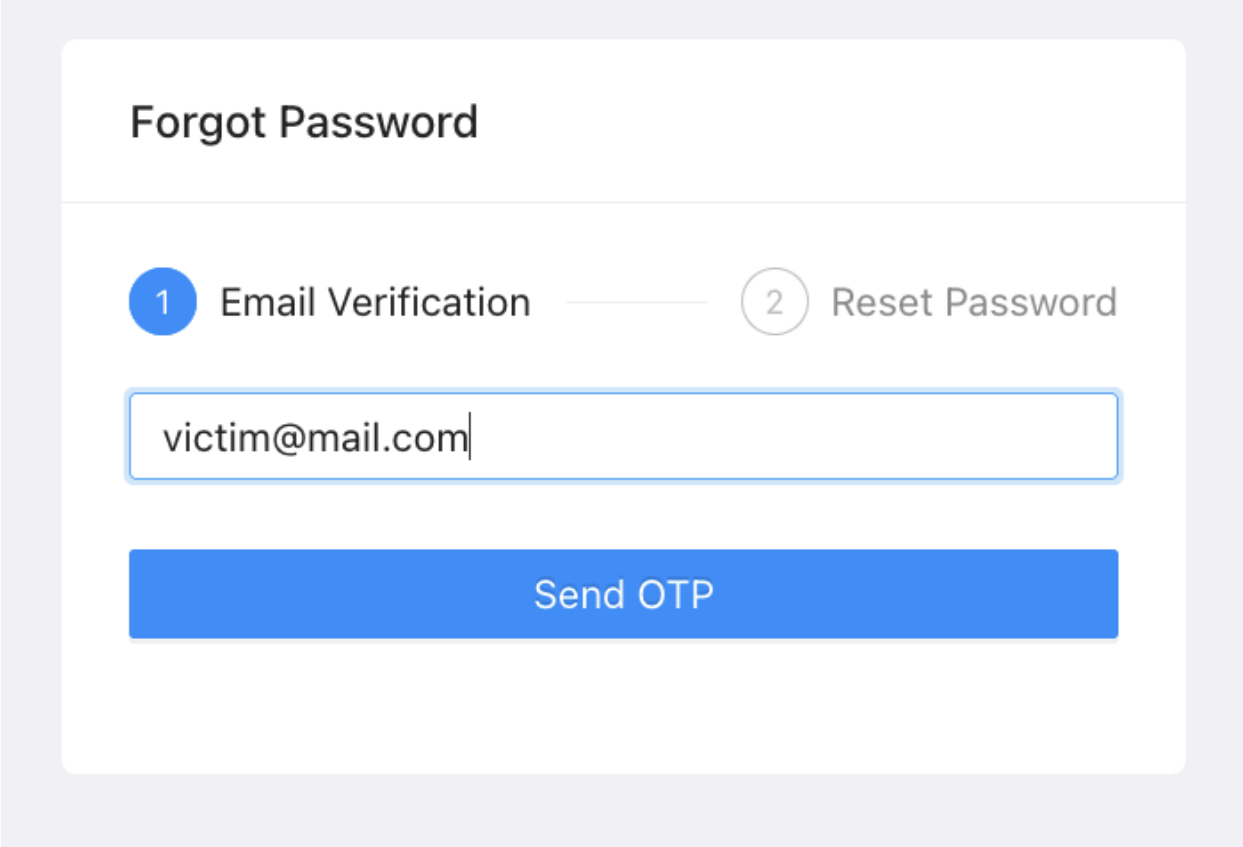
This is the post that is going to provide attacker the information to reset victim's own password.

4. Now go to the victim's post right-click → inspect → click the network tab → click on the request with URL: /post/{{id}} → click on the response section on the right side → see the details → attacker can get victim's email address from there.

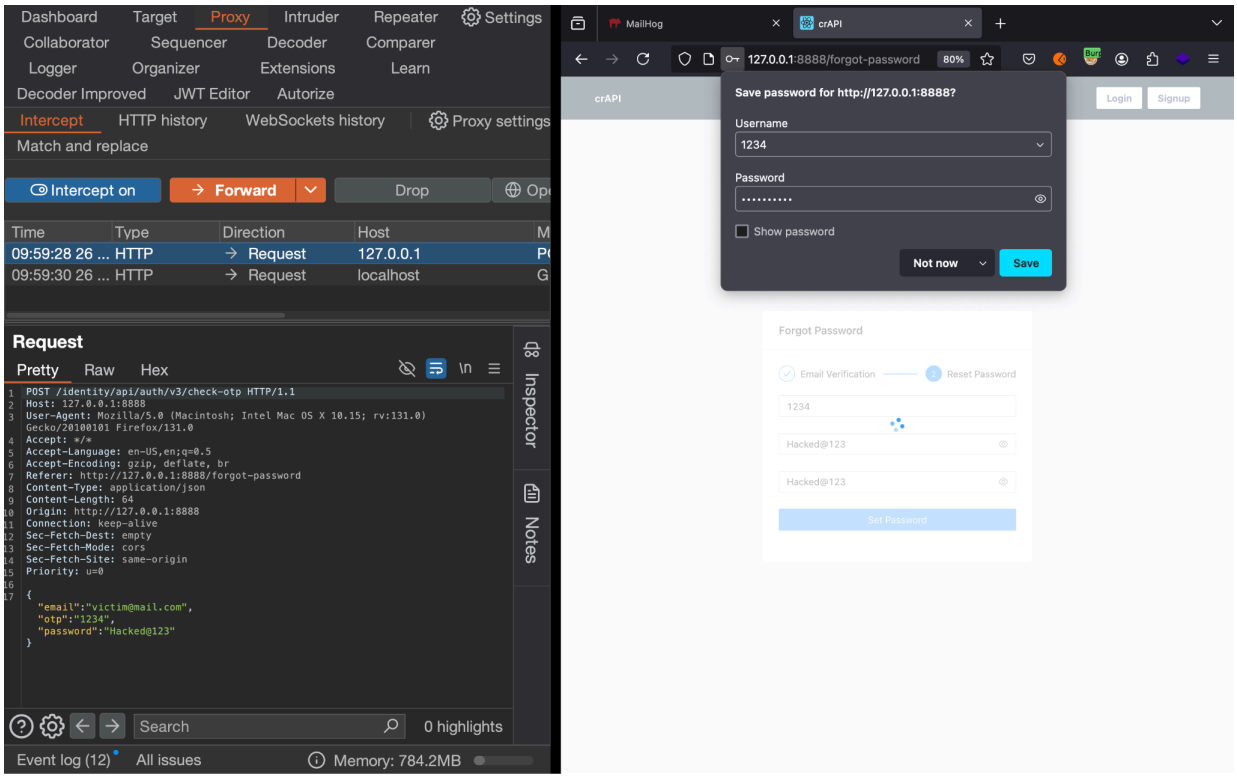


5. Now log-out → click on forgot password on login page

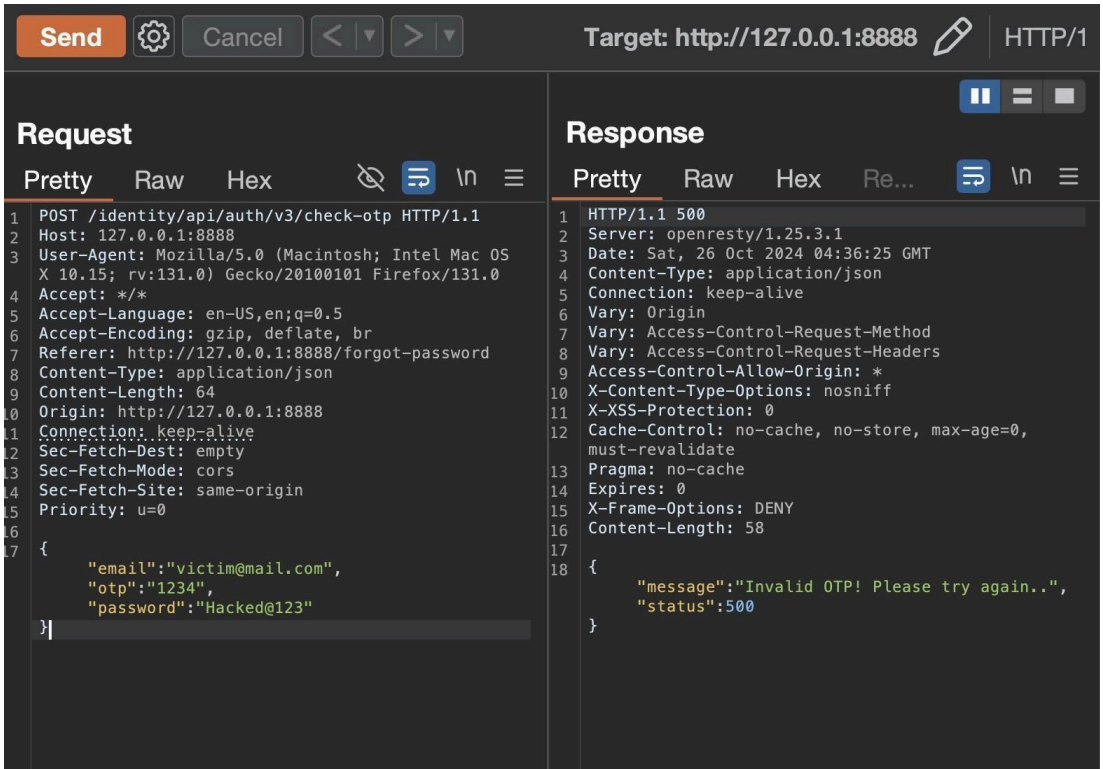
   → give victim's email address on email-verification box



6. Before clicking set password on the reset-password part, fire-up burp suite (assuming proxy settings are right) → click on intercept → then click the set password button



7. Right-click on the request → click send to repeater → click send → see the response with invalid otp



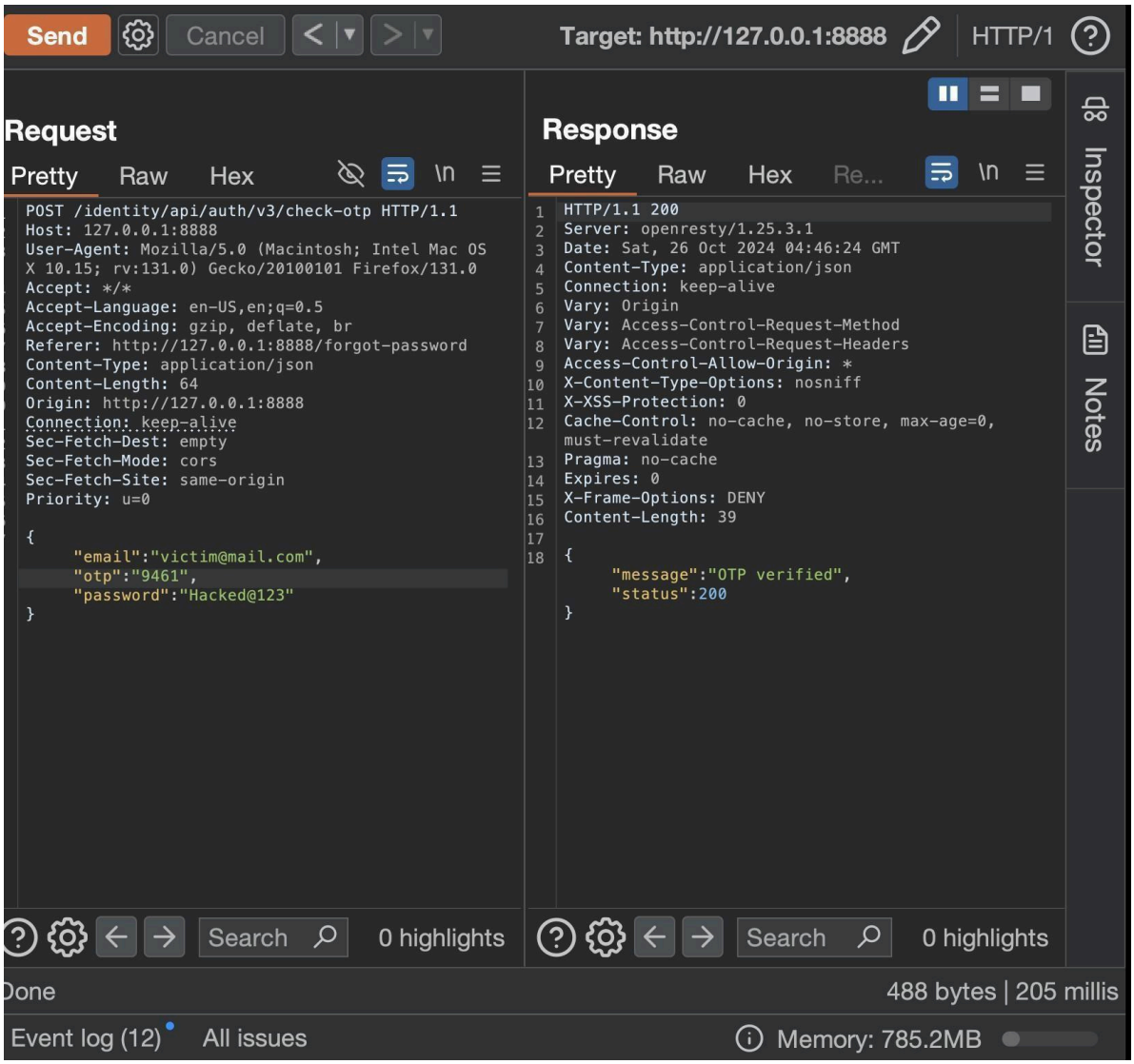
8. Let's try brute forcing the otp to see if it works, for that I'm using ffuf here as the burp's intruder is too slow → give this command in ffuf to get the otp.

Command:

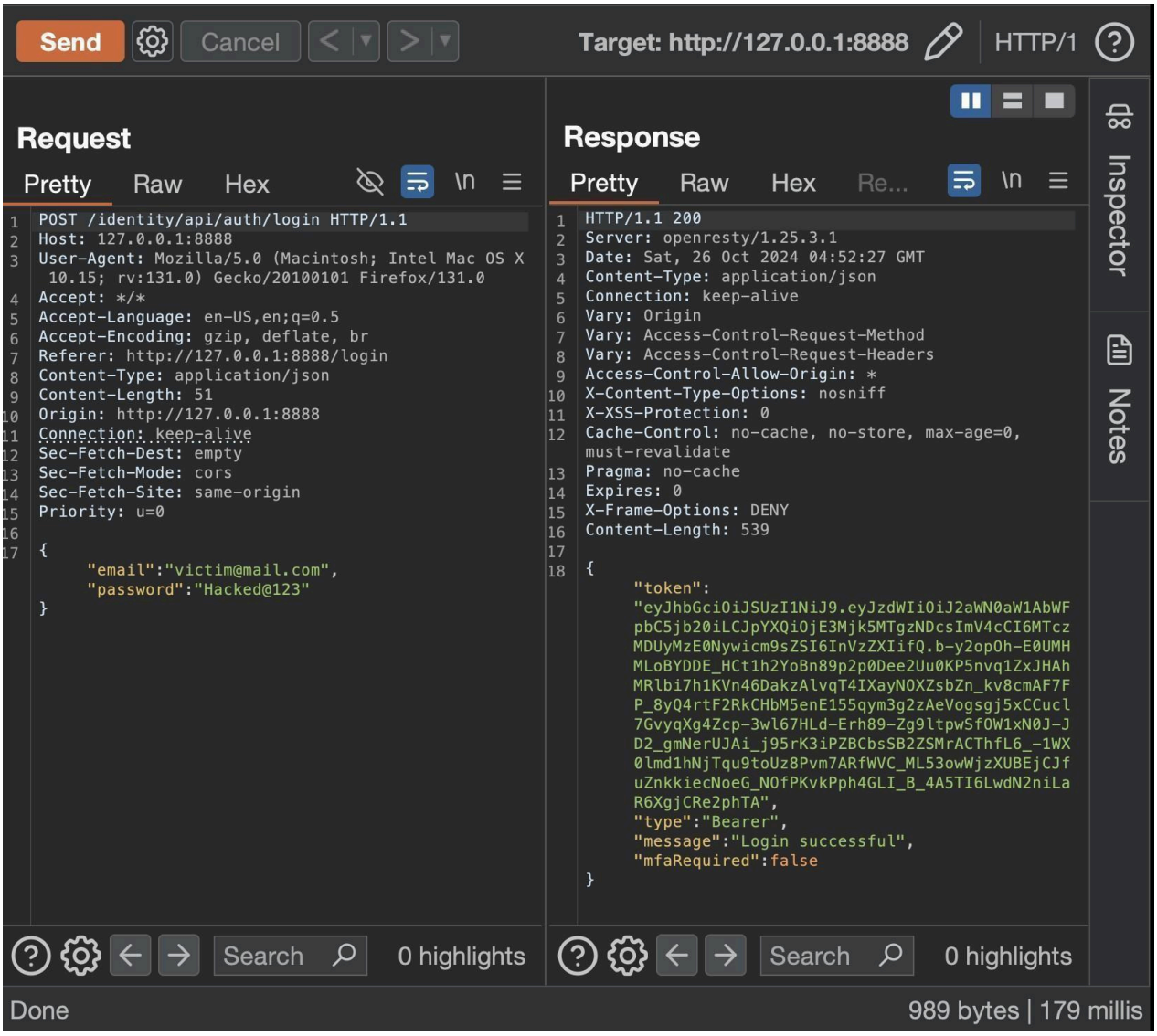
```
ffuf -w /Users/qatardoha/seclists/Fuzzing/4-digits-0000-9999.txt -u http: /127.0.0.1:8888/identity/api/auth/v2/check-otp -H "Content-Type: application/json" -X POST -d '{"email":"victim@mail.com","otp":"FUZZ","password":"Hacked@123"}' -mc 200
```

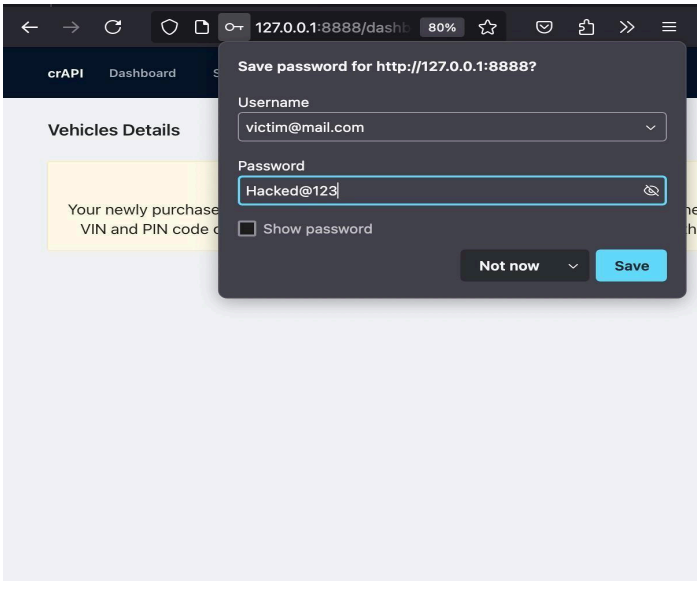
9. See we got the otp as '9461' Let's use this to login as the victim user.

10. See the response that we got in burp suite the password has successfully been reset.



11. See here I just logged in as victim user:





Impact Company &

Customer:

- This vulnerability can result in completely locking users out of there accounts
- Could change customer informations and impersonate them
- Financial loss as it may disrupt business flow

- Distrust From customers

- Repuational Damage

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- Violence of legal compliances

- Legal action from customer's side as the PII information may be leaked
- Legal action from authorities as it may violate IT laws of customer data protection.

Steps for mitigation

These are some of the counter measures to be taken for to get rid of these sort of vulnerabilities:

- In this specific vulnerability bruteforcing was the way that attacker got access into, so rate limiting the requests will help in defending against bruteforce

- API v3 was already immune to brute force by implementing rate limiting, attacker got arround of this protection mechanism is by simply changing the API version from v3 to v2 on the url, if the assets were properly managed then attacker wont be possible to access the v2 version.

- Keep documentation of every API version that company uses, and make sure the older versions are not accessible to the public, so that these sort of vulnerability where attacker can just change the version and get around the security mechanism may never happen.

- Use multifactor authentication, here to reset the password an otp is send to mail and using that we can reset the password, if we also enable a secondary mechanism to very the user it will be great

- otp was 4 digits numerical, make it 6 digit alphanumeric

Vulnerability no. 4

Name of Vulnerability: Excessive Data Exposure

- **Affected Component:** Recent post checking endpoint
- **Affected URL:**

http://127.0.0.1:8888/community/api/v2/community/posts/recent

Description Of Vulnerability:

This vulnerability occurs when the api request to an endpoint and returns information that shouldn't be accessible

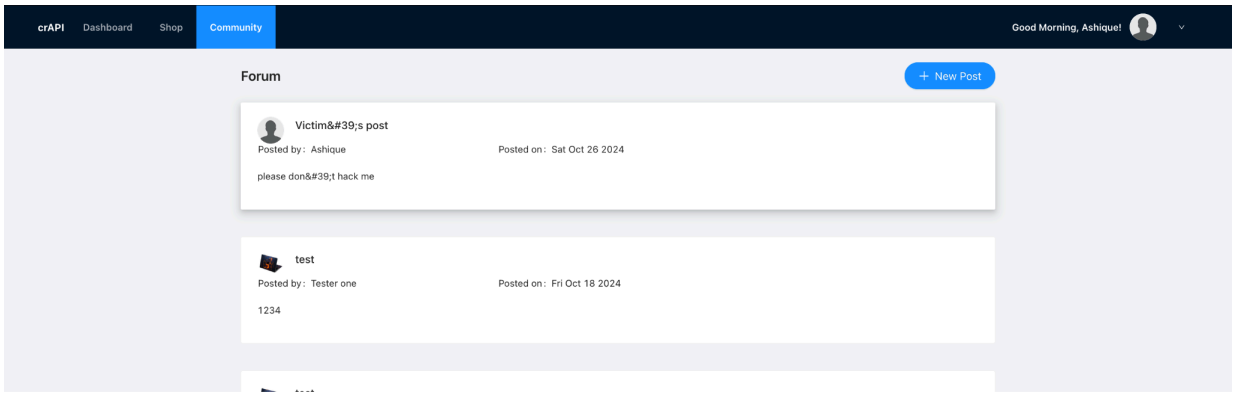
In this specific vulnerability, on the community forum we can see posts from different users, on inspecting more I found out that there is an api call made which contains /posts/recent on inspecting more, I found out that data of other users have also been available to me on the response.

Steps to Reproduce the vulnerability:

Tools used:

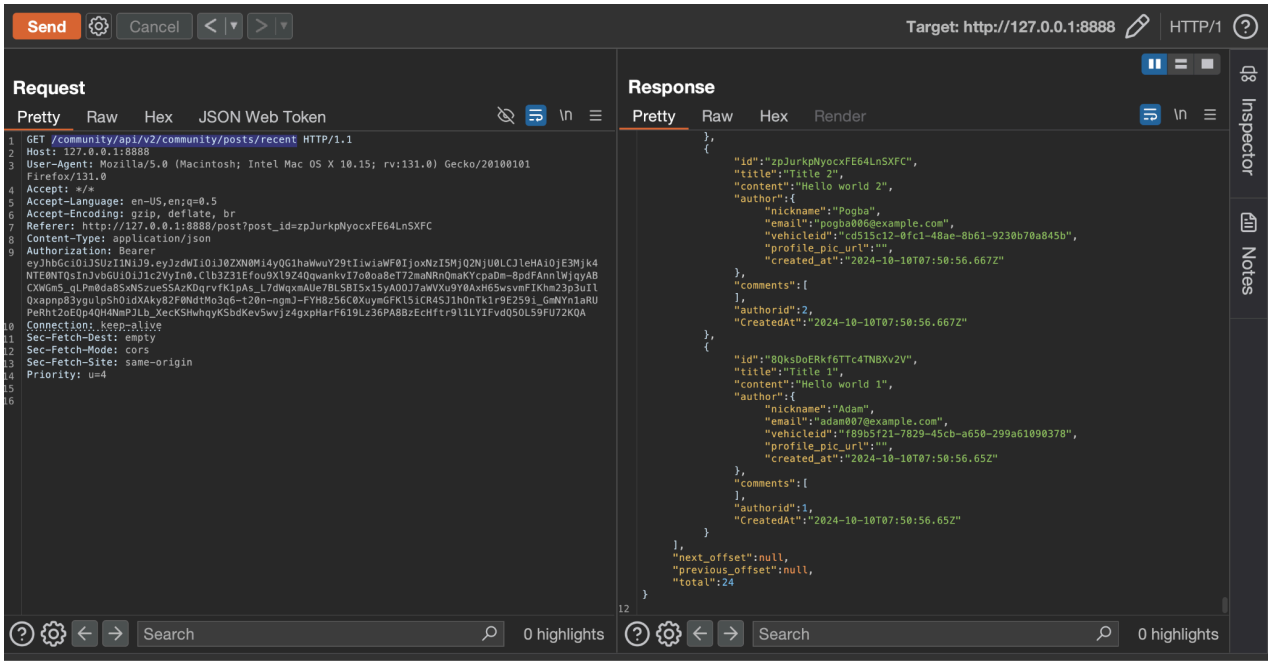
- Burp suite
- web inspector

1. Go to the community tab



2. Now open web inspector → click on network tab and hit refresh → see that there is a request with /posts/recent → intercept on burp suite → click the request and click resend
   → capture the request on burp and send it to repeater →

   click send



This is indeed a vulnerability the web app returns data of other users too in here.

Impact Company &

Customer:

1. Sensitive data of customers exposed may cause legal issues
2. Company may has reputational damage as this becomes a news

3. Financial and time loss for the company

4. Public data breaches erode trust, affecting customer relationships and brand value.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- Exposing sensitive data, like personal identifiers or financial information, can breach privacy laws (e.g., GDPR, CCPA, HIPAA).
- Non-compliance may result in hefty fines, varying by jurisdiction and severity of the exposure.
- Individuals affected by exposed data can sue, leading to costly settlements or damages.
- Authorities may conduct audits, impacting business operations and credibility.

Steps for mitigation

- Ensure the API only returns data that the authenticated user has permission to access. Implement strict filtering so that only necessary fields are returned.
- Enforce access control at the endpoint level. Use role-based access controls to ensure users can only access their own data or data they're authorized to view.
- Before sending the response, validate the data to ensure that no unintended fields are exposed. Use a whitelist approach, allowing only approved data fields to be returned.
- Apply masking for sensitive fields (e.g., emails, contact details) that should not be fully accessible to other users.
- API gateways can enforce access control policies and filter data responses. This adds an extra layer of security to prevent unauthorized data exposure.

Vulnerability no. 5

Name of Vulnerability: Excessive Data Exposure

- **Affected Component:** Video upload functionality

- **Affected URL:**

http://localhost:8888/identity/api/v2/user/videos

Description Of Vulnerability:

This vulnerability occurs when the api request to an endpoint and returns information that shouldn't be accessible or needed there.

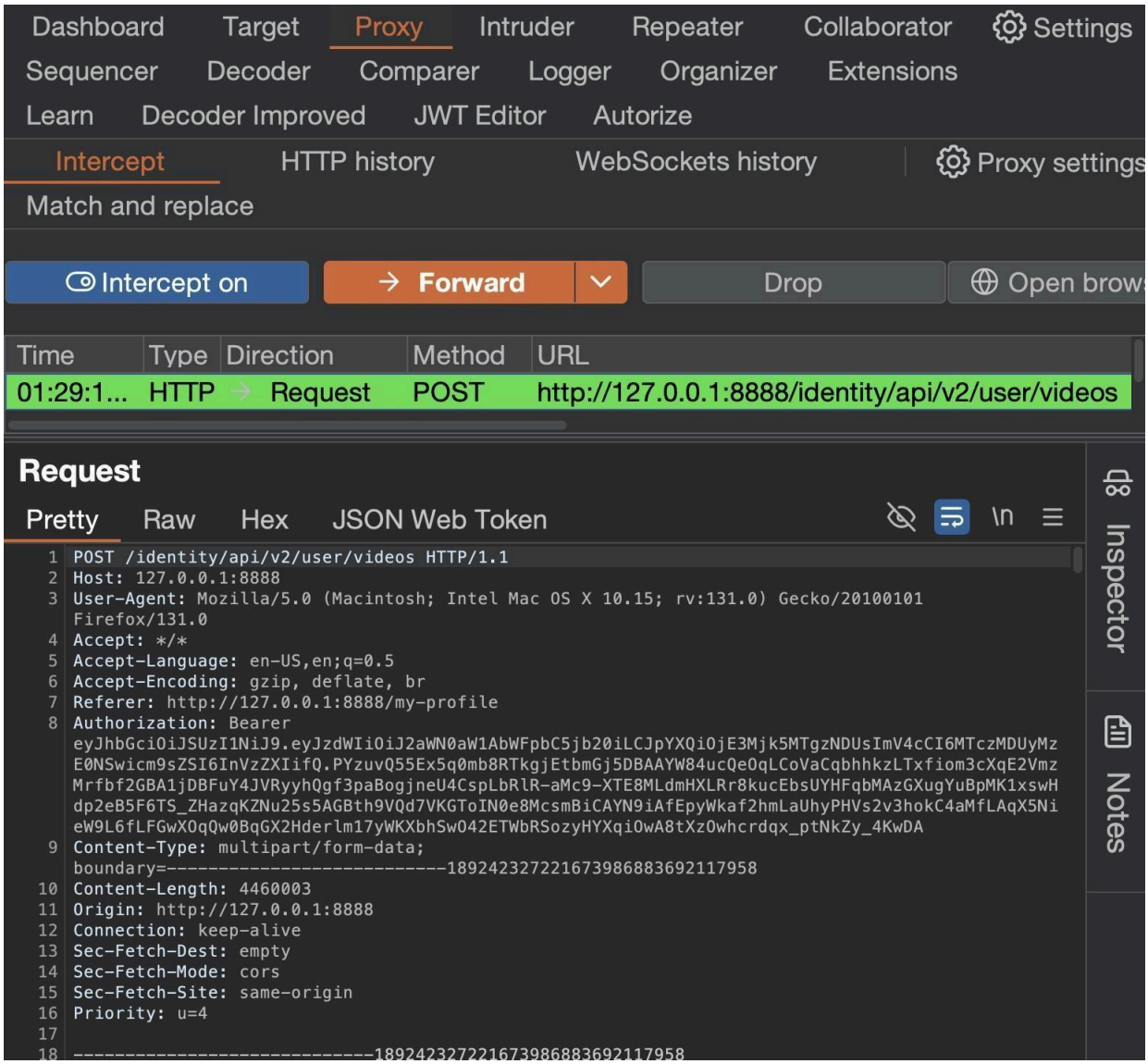
In this specific scenario, on the video upload functionality, there is an excessive data exposure, this may be used against the user in future and this might become key to a vulnerability

Steps to Reproduce the vulnerability:

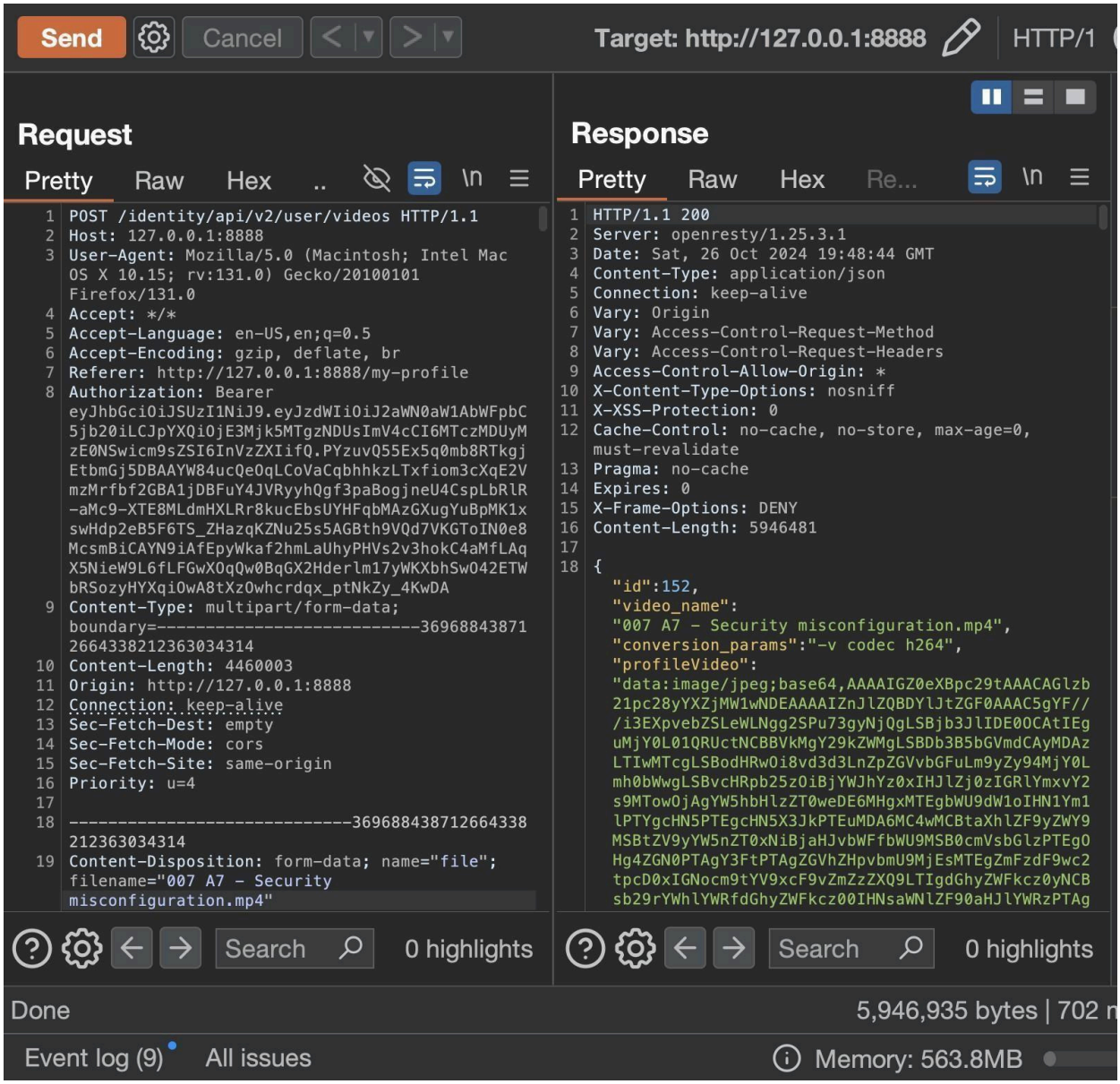
Tools used:

- Burp suite

1. Click on the profile picture → that will redirect you to profile page → click on the three dots and click upload video → before setting up the video, fire up burp suite and intercept the request like this:



2. Send this request to repeater and see on the response that some additional information can be seen



This disclosed internal properties of the video.

Impact

Company & Customer:

1. Sensitive data of customers exposed may cause legal issues
2. Company may has reputational damage as this becomes a news
3. Financial and time loss for the company
4. Public data breaches erode trust, affecting customer relationships and brand value.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- Exposing sensitive data, like personal identifiers or financial information, can breach privacy laws (e.g., GDPR, CCPA, HIPAA).
- Non-compliance may result in hefty fines, varying by jurisdiction and severity of the exposure.
- Individuals affected by exposed data can sue, leading to costly settlements or damages.
- Authorities may conduct audits, impacting business operations and credibility.

Steps for mitigation

- Ensure the API only returns data that the authenticated user has permission to access. Implement strict filtering so that only necessary fields are returned.

- Enforce access control at the endpoint level. Use role-based access controls to ensure users can only access their own data or data they're authorized to view.

- Before sending the response, validate the data to ensure that no unintended fields are exposed. Use a whitelist approach, allowing only approved data fields to be returned.

- Apply masking for sensitive fields (e.g., emails, contact details) that should not be fully accessible to other users.

- API gateways can enforce access control policies and filter data responses. This adds an extra layer of security to prevent unauthorized data exposure.

Vulnerability no. 6

Name of Vulnerability: **Rate limiting**

- **Affected Component: Contact Mechanic**

- **Affected URL:**

http://127.0.0.1:8888/contact-mechanic?VIN={{id}}

Description Of Vulnerability:

Rate limiting vulnerabilities happens when the application fails to limit the requests that come to the server maliciously, with an intent of DoS or so.

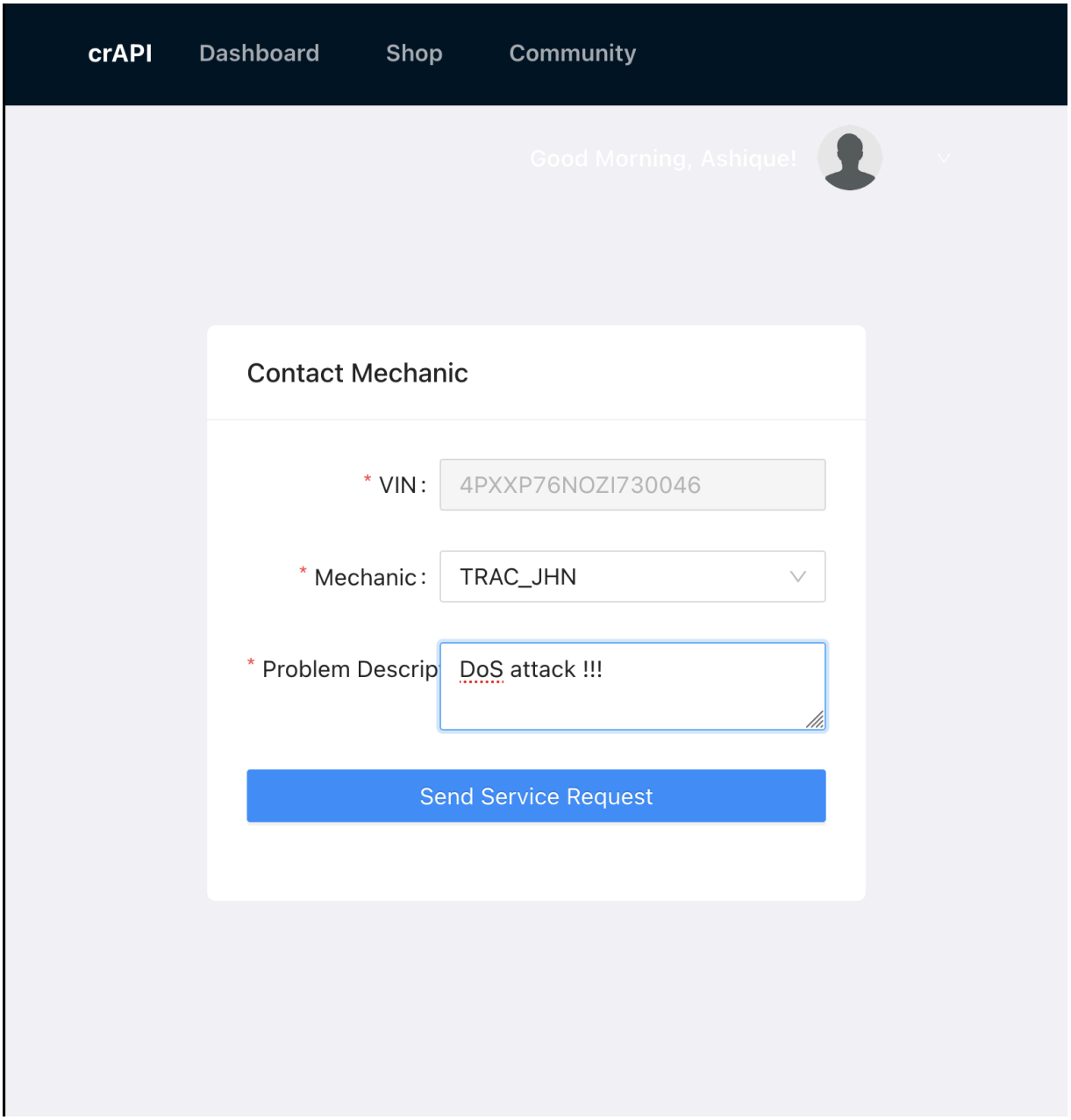
This specific vulnerability happens to the contact mechanic function where the user is able to send maximum number of requests, that causes to layer 7 DoS attack.

Steps to Reproduce the vulnerability:

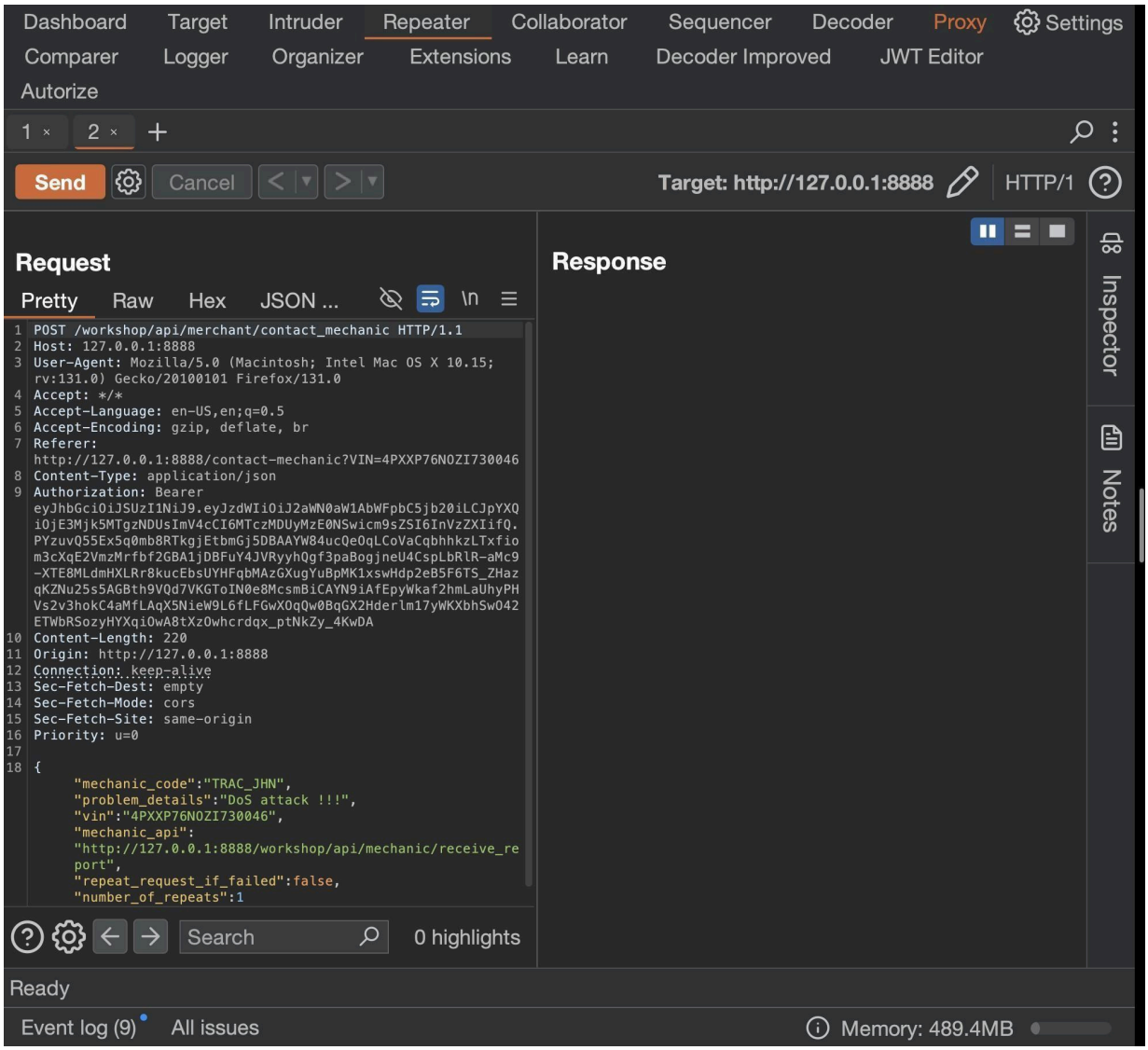
Tools Used:

- Burp suite

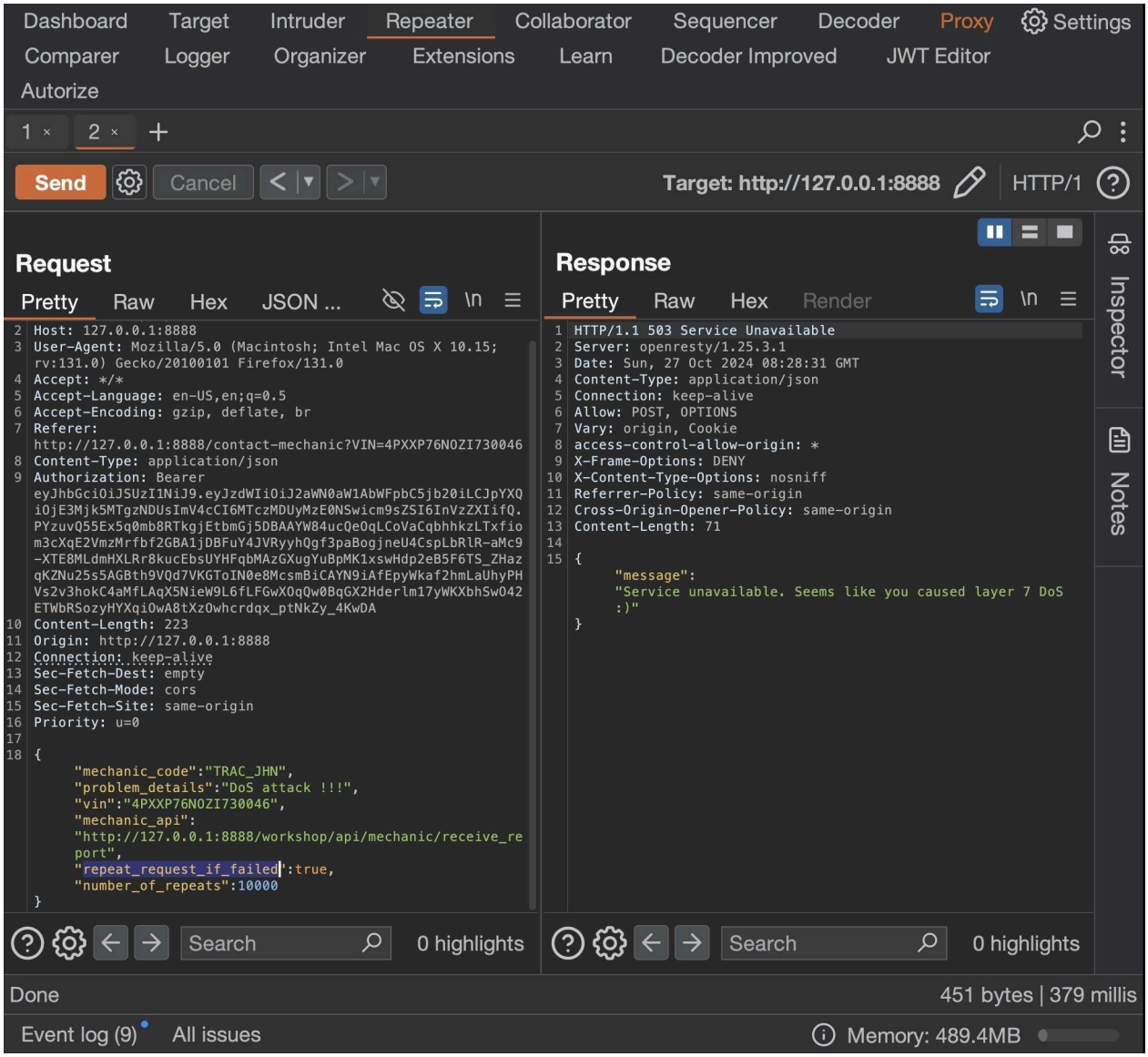
1. Go to the Dashboard → click contact mechanic → fill the details → before clicking send use burp to intercept.



2. Now Fire-up burp suite → intercept the request → send it to repeater



3. Now observe the value “number_of _requests” set it to 10000, also set “repeat_request_if_failed” to true now see the DoS attack is successful.



See the reponse indicate that layer 7 DoS attack is performed.

Impact

Company & Customer:

The failure to limit requests can lead to a layer 7 DoS attack, potentially bringing down critical customer-facing services. This can erode customer trust, leading to lost revenue and reputation damage.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- The absence of rate limiting could violate data protection and cybersecurity regulations (e.g., GDPR, CCPA), especially if downtime affects data access or critical services.
  Non-compliance may result in fines, sanctions, or lawsuits.

Steps for mitigation

- **Implement Rate Limiting**: Set request limits to a threshold that prevents abuse while allowing normal user activity.
- **Monitor and Block IPs**: Use tools to track IPs with excessive requests and block or throttle them to prevent repeated attacks.
- **Use CAPTCHA or MFA**: Apply these methods to restrict bot traffic on the contact form, adding a layer of user verification.
- **Optimize Server Resources**: Set automatic scaling to handle sudden traffic spikes without affecting application stability.

Vulnerability no. 7

Name of Vulnerability: **Broken Function Level Authorization**

- **Affected Component: video**

- **Affected**

http://127.0.0.1:8888/identity/api/v2/user/videos/152

Description Of Vulnerability:

This is similar to BOLA vulnerability that we encountered earlier, here the only difference is that the attacker get function level access to other users unauthorised, resulting in just deleting the account, updating data etc.

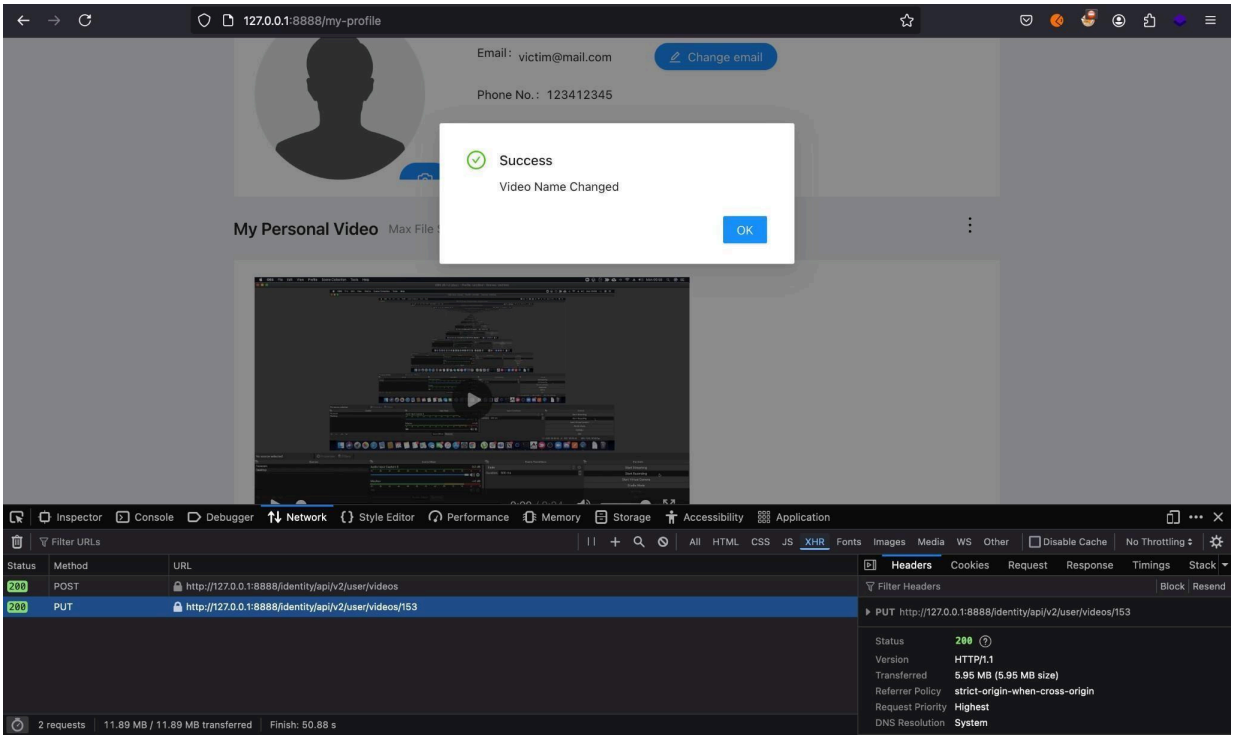
Here in this specific scenario i was able to access the admin endpoint to delete videos uploaded by another user.

Steps to Reproduce the vulnerability:

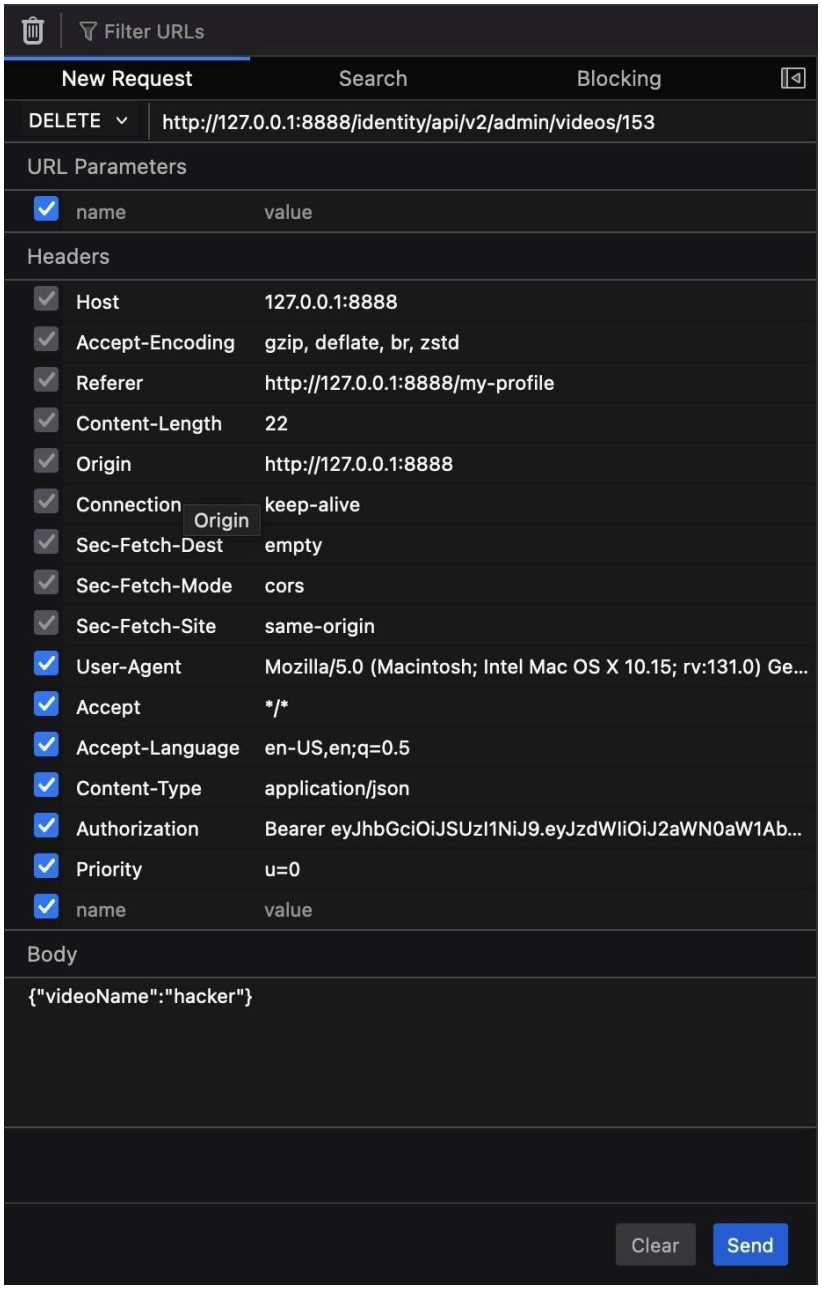
Tools Used:

- Web inspector

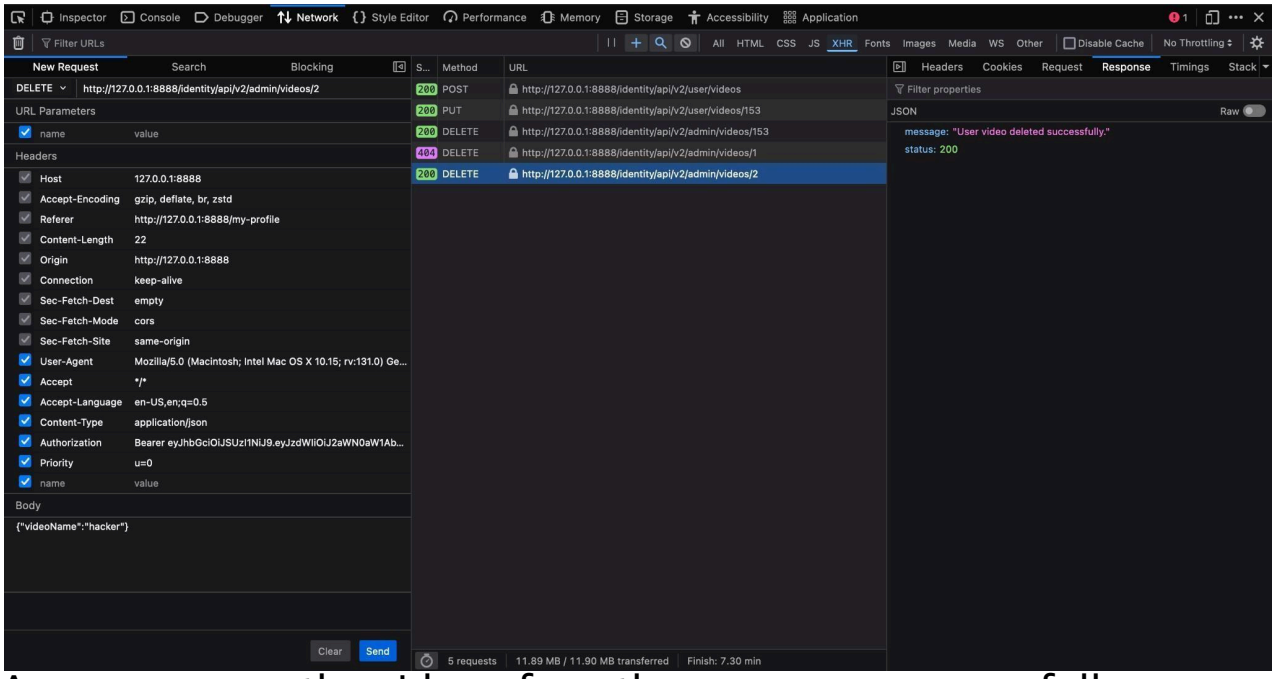
1. Go to the profile page → let's open web inspector → go to network tab → on uploading a video see that a request pops up → just ignore that and now change the video name see a request pop up



2. Now right click the request → click edit and resend → on experimenting i found out admin can delete video so let's try that, change the method to delete → change the user in url to admin.



3. By doing this it's possible to delete the video that we uploaded, changing the numeric value after /video will result in deleting video uploaded by another user.



As you can see the video of another user was successfully deleted by this method

Impact

Company & Customer:

This vulnerability can harm both company reputation and customer trust. Unauthorized access to critical functions, such as deleting users' uploaded content, can lead to user dissatisfaction, data loss, and potential financial losses. Customers may perceive the platform as insecure, possibly leading to reduced user engagement or customer churn.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Privacy Violations:** Accessing and modifying data without authorization breaches data protection laws, such as GDPR.
- **Liability for Damages:** If customer data is lost or altered, the company could face claims for damages.
- **Compliance Issues:** If regulations require strict access controls, this vulnerability may result in compliance failures, risking fines.
- **Potential for Fines and Penalties:** Regulatory bodies may impose fines, especially if the vulnerability is exploited.

Steps for mitigation

- **Implement Strong Access Controls:** Use

role-based access control (RBAC) to restrict endpoint access based on user privileges.

- **Apply Function-Level Authorization Checks:** Ensure that every API endpoint includes authorization checks to validate user roles.

- **Limit Administrative Privileges:** Minimize access to admin-level endpoints to only those who need it, using multi-factor authentication.

- **Log and Monitor Access Patterns:** Regularly monitor access logs for unusual activity on sensitive endpoints.

- **Conduct Regular Penetration Testing:** Test for potential BOLA-like vulnerabilities on critical functions to identify and resolve gaps.

Vulnerability no. 8

Name of Vulnerability: Mass Assignment

- **Affected Component: Order Return**

- **Affected URL:** http://127.0.0.1:8888/past-orders

Description Of Vulnerability:

This is a vulnerability where an attacker can manipulate input data to modify an object's properties, often leading to unauthorized changes in a system

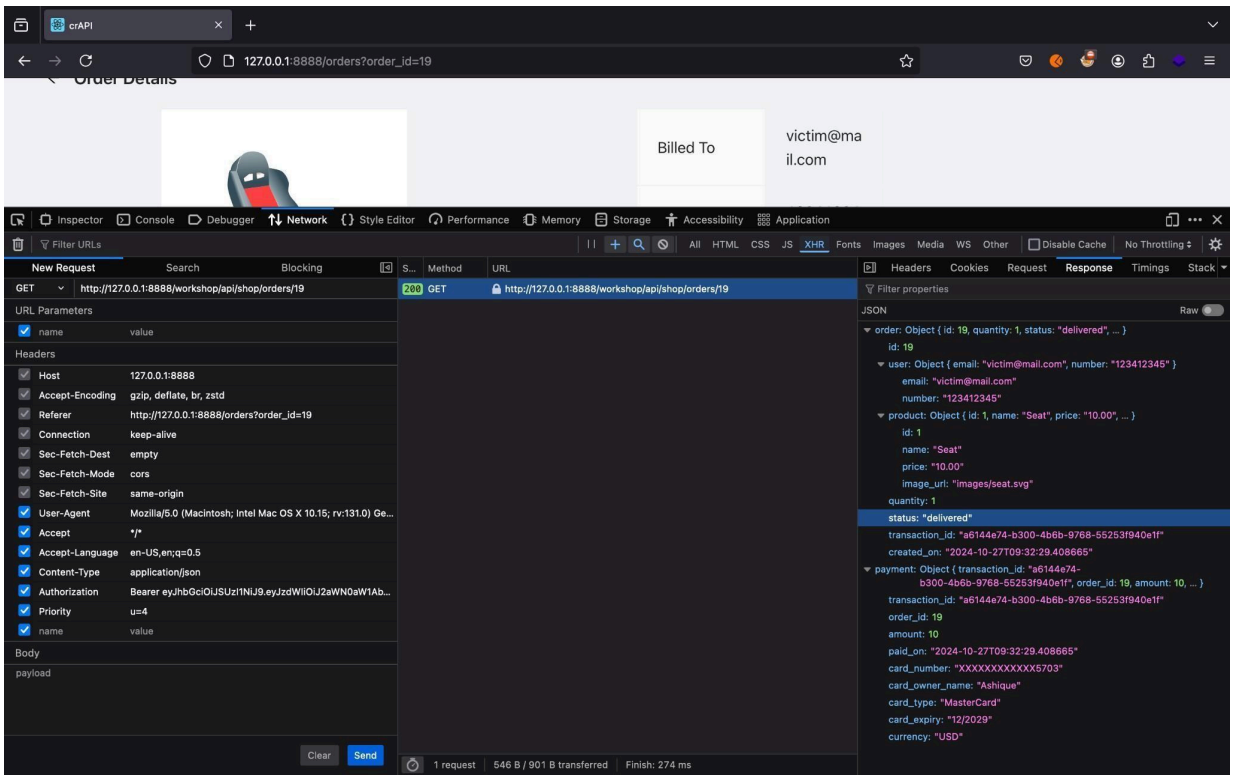
Here in this specific scenario, an attacker can order any product and to get the item for free he can keep an make the order status explicitly to returned and basically get the item for free.

Steps to Reproduce the vulnerability:

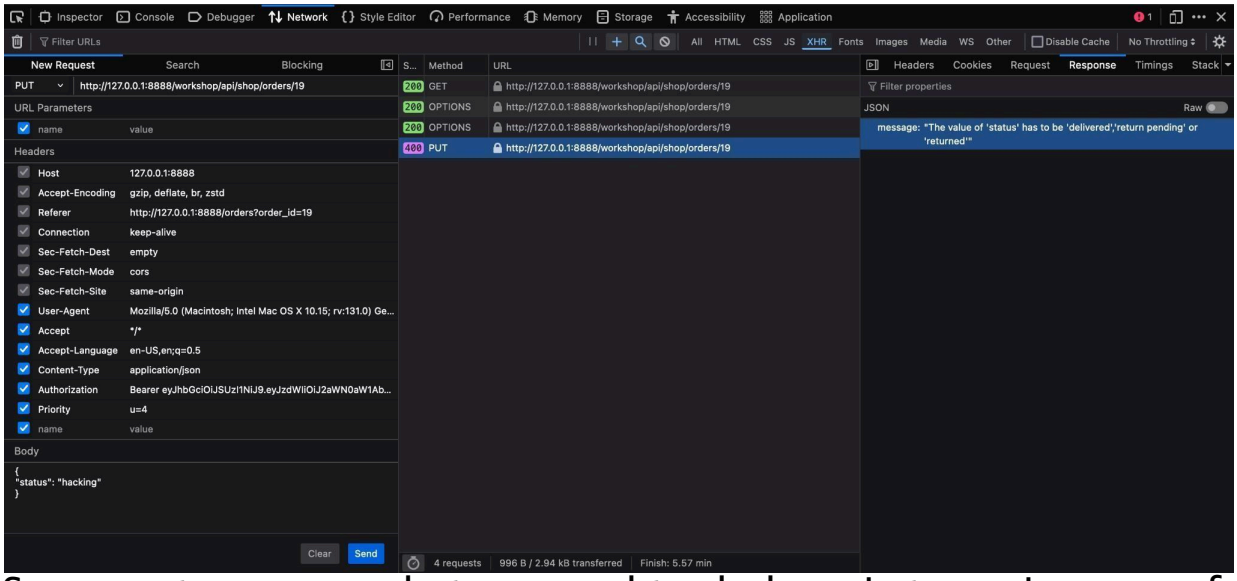
Tools Used:

- Web inspector

1. Go to shop → buy a product → load web inspector → go to network tab → click on order details → click on edit and resend

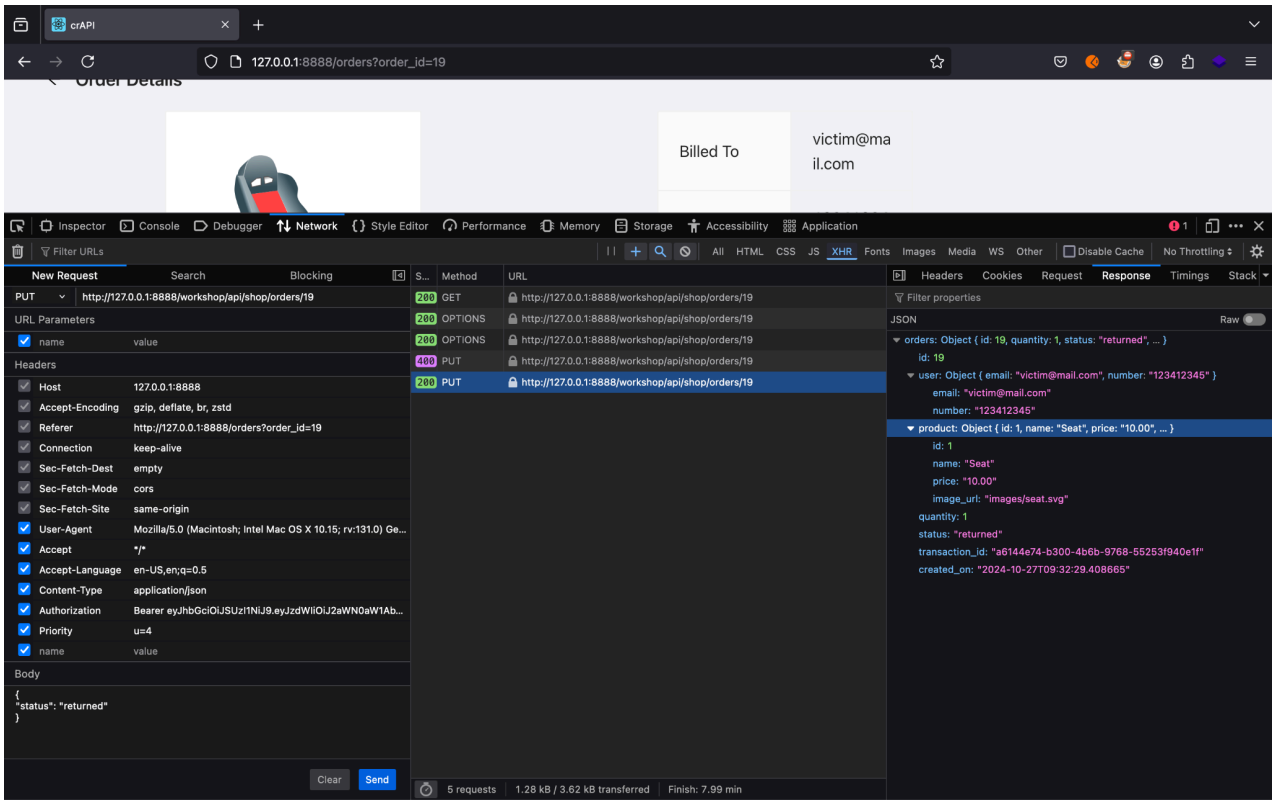


2. On experimenting i found that, all HTTP methods are allowed, so I tried PUT method, on the response of first request i saw a json data like this: status : “delivered” let me try including the data into the request and modify its value



So we got an error what we need to do here is to assign one of these values to the data.

3. So we need give one of these values, if we can set the value to returned then we get to keep the product without actually returning it.



Here you can see that the product is set as returned this is indeed a mass assignment vulnerability.

Impact

Company & Customer:

This vulnerability can lead to direct financial loss for the company as attackers exploit it to obtain products for free. It undermines the company's revenue and inventory management, creating potential distrust among legitimate customers and impacting overall customer experience.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Fraud and Theft:** Manipulating order statuses for free items may qualify as theft, leading to legal liabilities.
- **Compliance and Security Violations:** Lax input validation on order status could result in non-compliance with e-commerce security standards.
- **Risk of Financial Damages:** Financial impacts from exploited vulnerabilities could lead to investor or stakeholder dissatisfaction.

- **Contractual Breach:** If promised security standards are compromised, it may violate agreements with payment processors or customers.
- **Potential for Fines and Penalties:** Authorities may impose penalties if this vulnerability breaches anti-fraud regulations.

Steps for mitigation

- Implement Strong Input Validation: Restrict input fields to valid, allowed values and prevent unauthorized status updates.
- Verify User Roles and Permissions: Ensure only authorized users or system processes can change sensitive order statuses.
- Audit Order Status Changes: Keep logs of all order status modifications and monitor for unusual patterns or repetitive status changes.
- Secure API Endpoints: Protect endpoints that handle order details and status updates with strict access controls.
- Conduct Regular Security Audits: Regularly test and review code for vulnerabilities in data input and access handling.

Vulnerability no. 9

Name of Vulnerability: Mass Assignment

- **Affected Component: order return**

- **Affected URL:** http://127.0.0.1:8888/orders

Description Of Vulnerability:

This is a vulnerability where an attacker can manipulate input data to modify an object's properties, often leading to unauthorized changes in a system

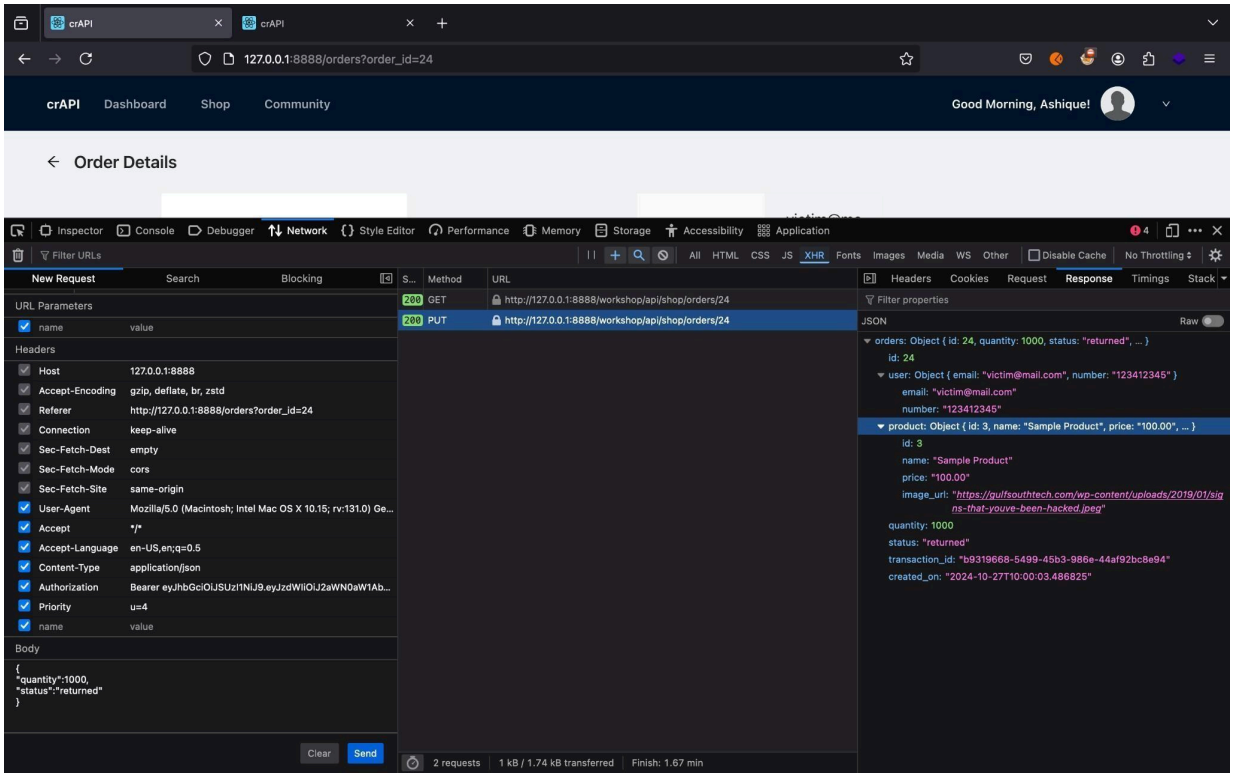
This specific vulnerability occurs when we can return any volume of products, which the attacker hasn't even ordered. This mass assignment can be considered as an extension of the vulnerability that we discussed earlier.

Steps to Reproduce the vulnerability:

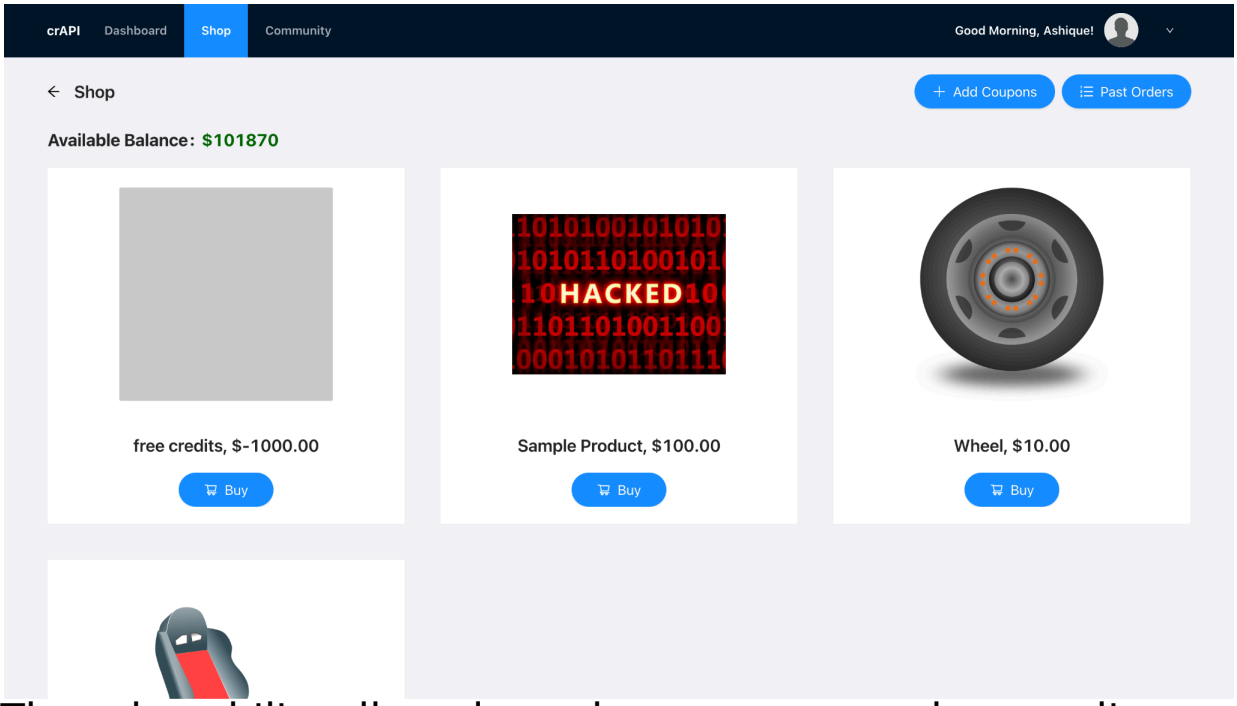
Tools Used:

- Web inspector

1. Go to the shop → click on order details, → just do as we did in vulnerability 8 but also include a field where the quantity can also be set like this:



2. Now see the credits:



The vulnerability allowed attacker to get as much as credits as he wants

Impact

Company & Customer:

This vulnerability can result in significant financial and operational losses. By exploiting the ability to return products that were never ordered, attackers can trigger false returns, leading to revenue loss and skewed inventory data. Such exploitation may cause stock inaccuracies, delivery delays, and erode customer trust.

Legal Issues:

Several legal concerns can arise, such as:

- **Fraud and Inventory Manipulation:**
  Fraudulently returning unpurchased items is theft and could lead to legal actions.
- **Non-Compliance with Security Standards:**
  Failure to enforce input controls and object permissions may breach security and e-commerce regulations.
- **Liability for Financial Losses:**
  Significant financial losses can lead to liability and possible litigation from stakeholders.

- **Contractual Breach:** Vulnerability could lead to breach of service contracts or agreements with vendors and partners.
- **Possible Regulatory Fines:** If this vulnerability impacts financial records, it may attract penalties from financial authorities.

Steps for Mitigation

- **Enforce Strict Input Validation:** Allow only legitimate values for return requests and verify product ownership.
- **Apply Mass Assignment Protections:** Prevent unauthorized data assignments by limiting object properties to expected values.
- **Implement Return Verification Checks:** Verify that the product was purchased before approving returns to prevent fraud.
- **Monitor and Audit Returns:** Track and flag irregular return patterns for review, and log return requests with customer details.

Vulnerability no. 10

Name of Vulnerability: **Mass Assignment**

- **Affected Component: video**

- **Affected URL:**

http://127.0.0.1:8888/identity/api/v2/user/videos/202

Description Of Vulnerability:

This is a vulnerability where an attacker can manipulate input data to modify an object's properties, often leading to unauthorized changes in a system

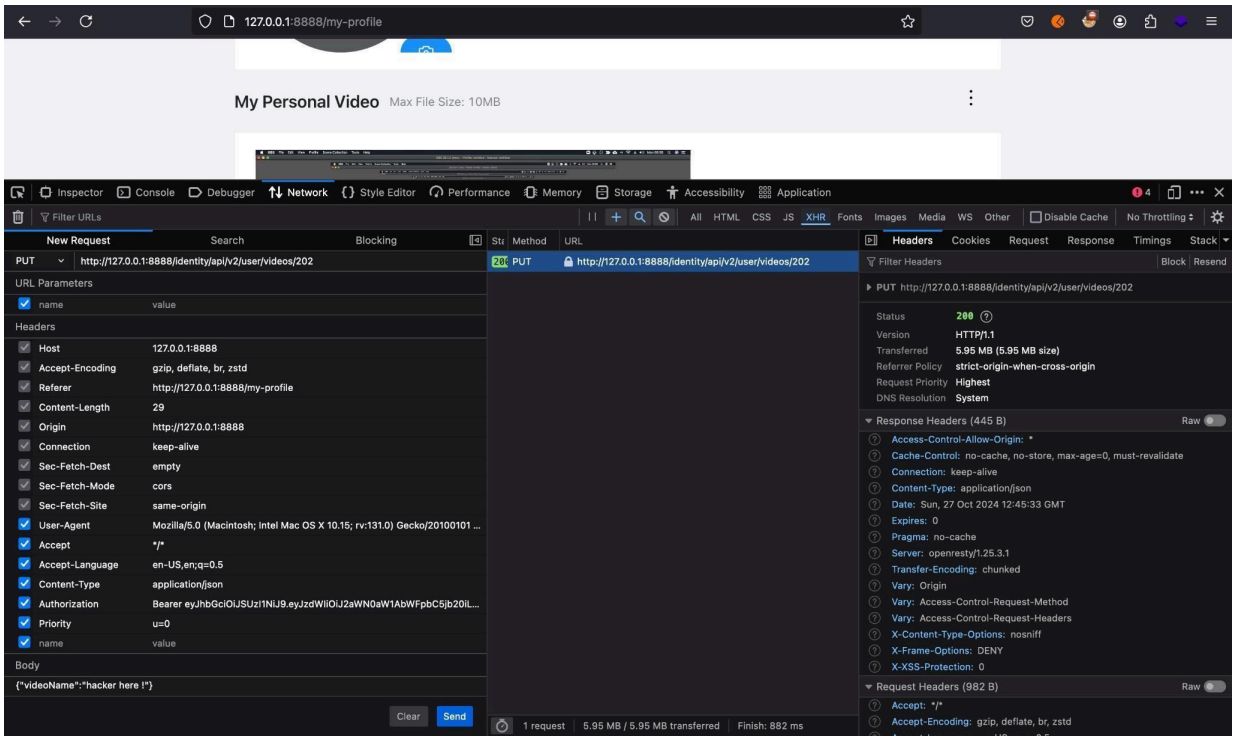
Here in this specific vulnerability I was able to change the internal property of a video just by assigning a PUT request to the endpoint.

Steps to Reproduce the vulnerability:

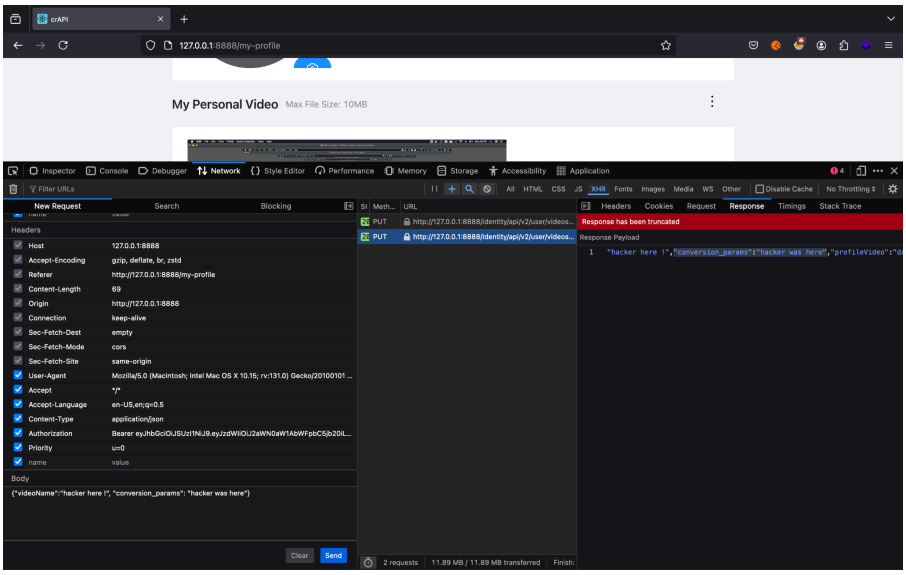
Tools Used:

- Web Inspector

1. Go to the profile page → bring up web inspector → go to network tab → rename the uploaded video on the page → see a request popping up with a video name parameter



2. On the response we can see "conversion_params":"-v codec h2" Let's try editing this to "conversion_params":"hacker was here" Let's see if it actually works



Impact

Company & Customer:

This vulnerability can lead to unauthorized data manipulation, which could damage the company's credibility and compromise data integrity. Attackers modifying video properties, such as access level or visibility, could expose private content, mislead users, or alter content categorization, impacting user experience and trust.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Privacy Violations:** Unauthorized property changes, such as making private videos public, may breach privacy regulations.

- **Data Integrity and Compliance Risks:** Tampering with content data can lead to non-compliance with data integrity and security standards.
- **Liability for Unauthorized Changes:** If user data is altered without consent, the company may face claims or regulatory actions.
- **Breach of Service Agreements:** If data changes violate usage policies, it could result in contractual breaches.
- **Penalties for Regulatory Non-Compliance:** Exposure of restricted content may attract penalties, especially if it affects regulated content.

Steps for mitigation

- Enforce Proper Access Control: Ensure only authorized users can update specific properties, and limit permissions for sensitive data.
- Validate Input Data: Filter and validate input to prevent unauthorized property changes on sensitive objects.

- Implement Object-Level Authorization: Ensure proper permissions are enforced at the object level before processing requests.
- Log and Monitor Changes: Track changes made via PUT requests, particularly on sensitive endpoints, and flag unusual activity.
- Conduct Routine Security Audits: Regularly review and test endpoints to detect and resolve vulnerabilities related to object manipulation.

Vulnerability no. 11

Name of Vulnerability: SSRF

- **Affected Component: Contact Mechanic**

- **Affected URL:** http://127.0.0.1:8888/contact-mechanic

Description Of Vulnerability:

SSRF is a type of security vulnerability that occurs when an attacker can manipulate the requests made by a web application to access resources on the server or other internal systems that they should not have access to.

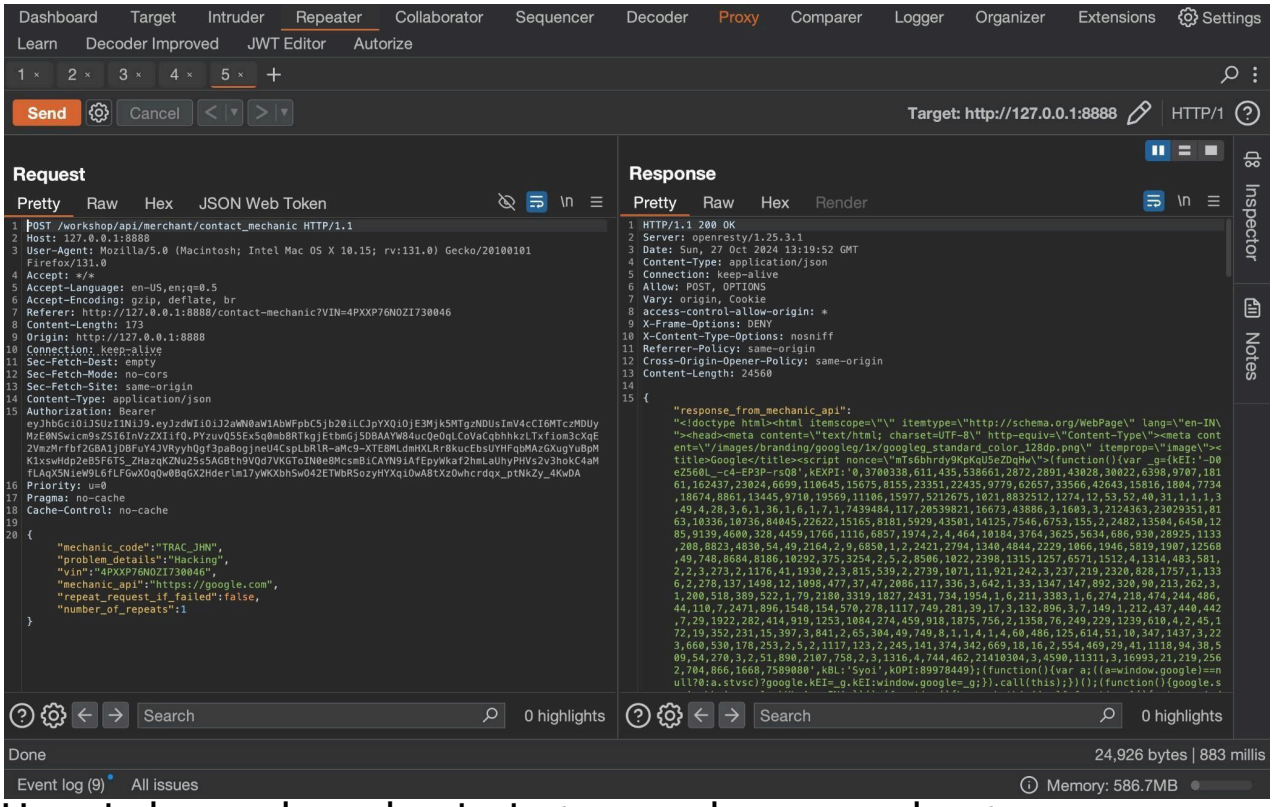
Here, in this vulnerability i was able to send a request to google.com from the part of the web application, this will result in api calls to any website from the web application.

Steps to Reproduce the vulnerability:

Tools Used:

- Burp suite

1. Go to the dashboard → click on contact mechanic → fill the details → intercept the request in burp suite → send it repeater



Here i changed mechanic_ip to google.com and got a response.

Impact Company &

Customer:

An SSRF vulnerability allows attackers to leverage the server to make unauthorized requests, which can lead to exposing sensitive internal systems and data. Attackers can access restricted services, potentially uncovering internal IPs, open ports, or sensitive endpoints, posing a risk to both the company and customer security.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Data Privacy Breach:** If internal systems or customer data are exposed, it may violate privacy regulations.
- **Non-Compliance with Security Standards:** SSRF vulnerabilities often lead to breaches in compliance with industry security standards.

- **Liability for Third-Party Damage:** If SSRF requests impact external services or APIs, the company may face legal claims.
- **Potential Regulatory Fines:** Exploitation leading to data exposure or unauthorized access may incur fines from regulatory bodies.

Steps for mitigation

- Restrict Outbound Requests: Limit requests from the application to trusted and necessary external domains.
- Validate User Input Strictly: Ensure URL parameters are validated, whitelisting only trusted domains.
- Enforce Network Access Controls: Prevent the web application from accessing internal or restricted network segments.
- Log and Monitor Outgoing Requests: Track and review outgoing requests for suspicious activity to detect SSRF attempts.

Vulnerability no. 12

Name of Vulnerability: NoSQL Injection

- **Affected Component: coupon**

- **Affected URL:**

  http://127.0.0.1:8888/community/api/v2/coupon/validate- coupon

Description Of Vulnerability:

NoSQL Injection is a type of security vulnerability that occurs in applications that use NoSQL databases, such as MongoDB when user-supplied data is not properly sanitized or validated before being used in database queries.

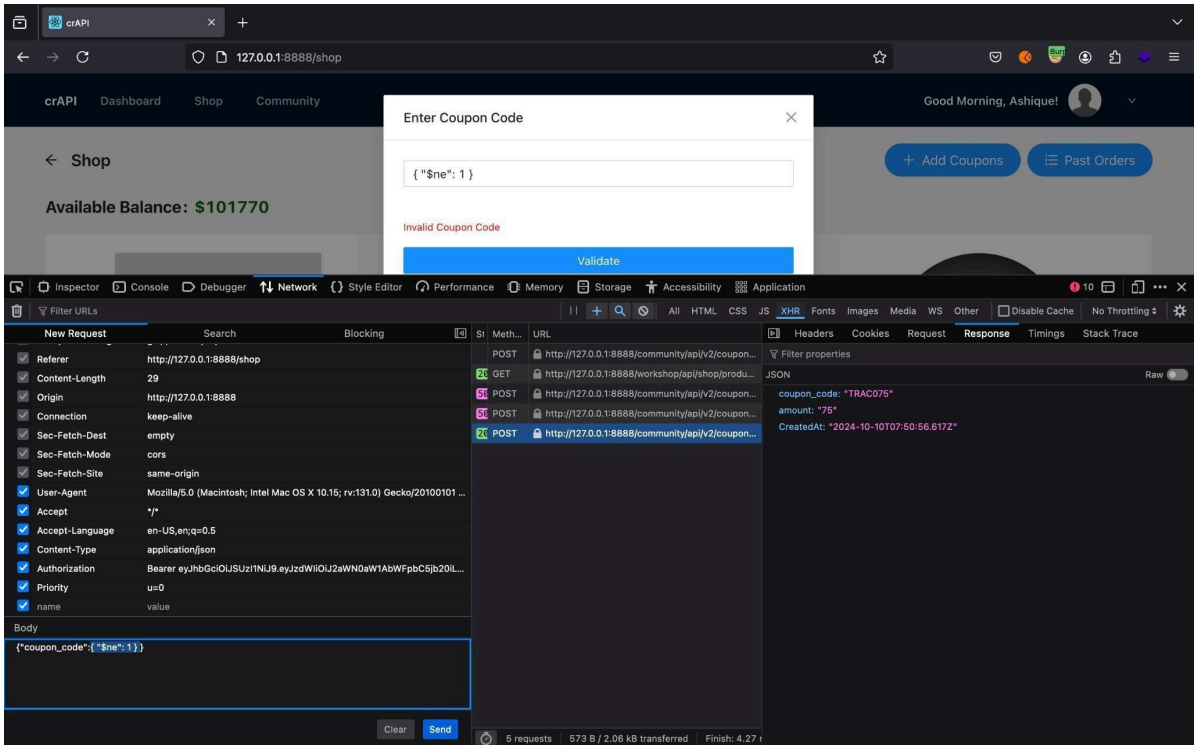
Here in this specific vulnerability i tried a NoSQL payload on the coupon code and it worked

Steps to Reproduce the vulnerability:

Tools Used:

- Web inspector

1. Click on the shop tab → click on coupon → open web inspector → click to validate the coupon code → click edit and resend the request → Give this payload instead the real value: { "$ne": 1 }. This will activate coupon



Impact Company &

Customer:

A NoSQL injection vulnerability, especially in coupon code fields, can allow attackers to bypass coupon restrictions, potentially redeeming unauthorized discounts, causing financial loss for the company, and leading to an unfair user experience. Customers may lose trust in the platform if exploitation impacts their transactions or personal data.

Exploiting coupon fields can disrupt pricing models and financial forecasts. Attackers could create significant revenue losses by manipulating coupon discounts, affecting the company's revenue flow and leading to inaccurate accounting data.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Compliance Violations:** Poor input validation on database queries may breach security and data protection regulations.

- **Fraud and Financial Loss:** Exploiting coupon codes could be legally classified as fraud, leading to financial and reputational liabilities.
- **Contractual Breach:** If coupon abuse affects revenue, it may violate agreements with partners or stakeholders.
- **Risk of Penalties:** Regulators may impose penalties if customer data or finances are impacted by exploitation.

Steps for mitigation

- Sanitize and Validate Input: Enforce strict validation on coupon fields to prevent injection attacks.
- Use Parameterized Queries: In NoSQL databases, use parameterized or safe query-building techniques to prevent injection.
- Restrict Database Privileges: Limit permissions on coupon-related collections to minimize the impact of potential attacks.
- Log and Monitor Coupon Redemptions: Track unusual coupon usage patterns to detect possible injection attempts.

Vulnerability no. 13

Name of Vulnerability: **Unauthenticated access**

- **Affected Component: order Details**

- **Affected URL:**

http://127.0.0.1:8888/workshop/api/shop/orders/19

Description Of Vulnerability:

This refers to allowing users or clients to interact with a system or application without requiring them to provide any form of authentication or identification. This means that users can access certain resources or perform certain actions without needing to log in or provide credentials.

Here even though JWT token is not provided the request is accepted and we get a response from the web app this is indeed a vulnerability.

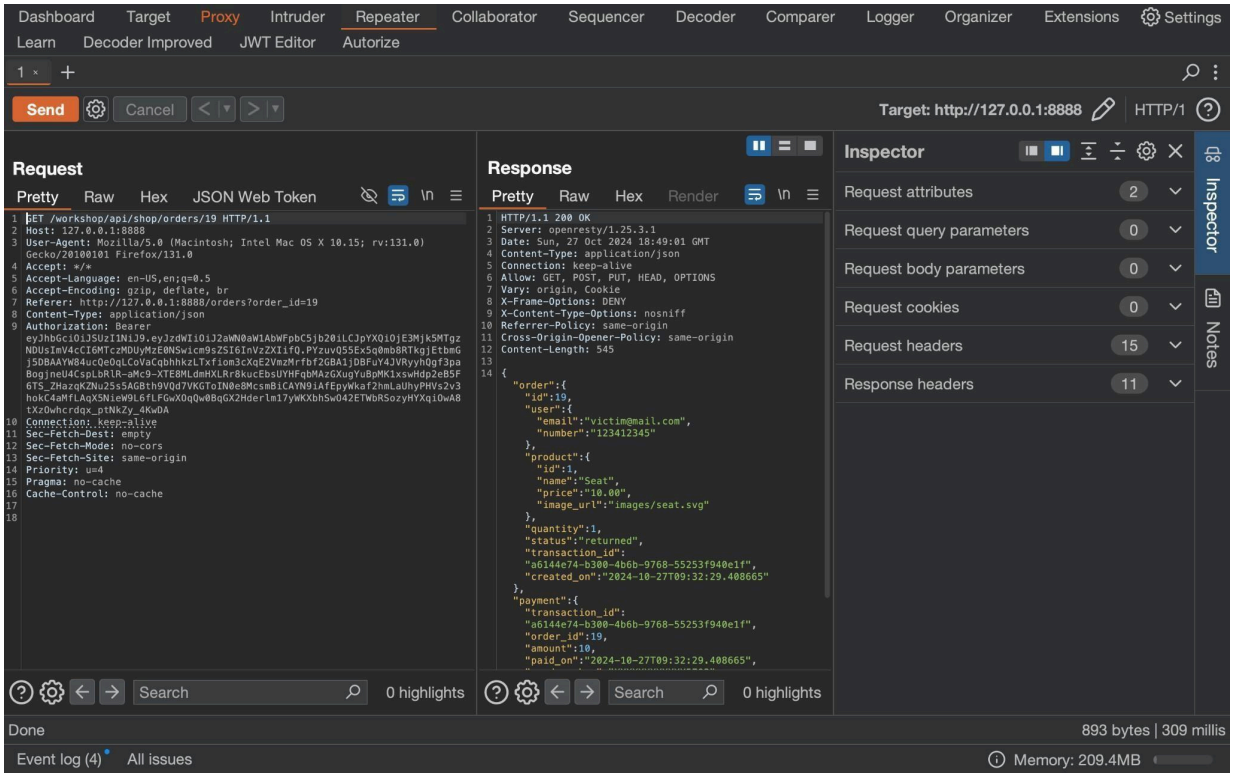
Steps to Reproduce the vulnerability:

Tools Used:

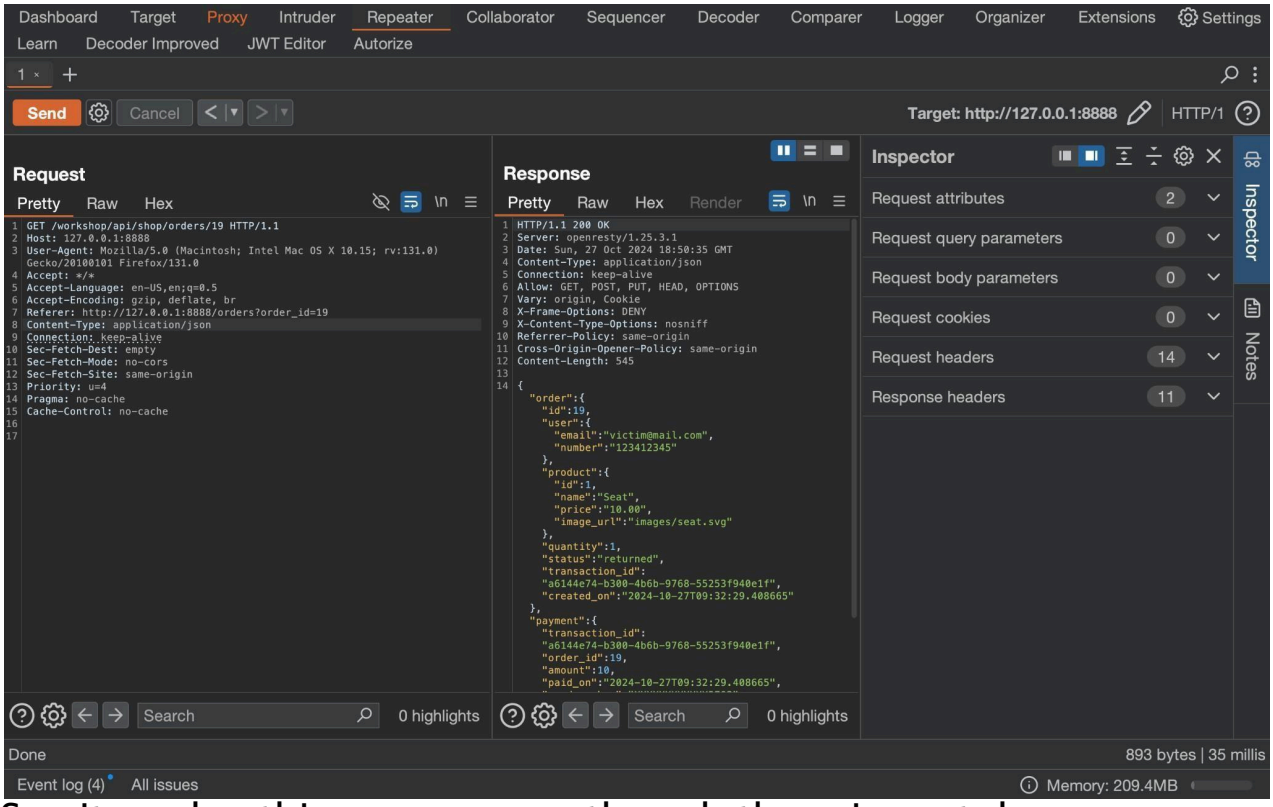
- Burp suite

1. Go to shop → click past orders → click order details → send the request to burp suite

Request with bearer token (JWT):



2. Now try removing the token and send the request again see what happens



See it works, this means even though there is no token we can get the web app to provide us with resources we ask.

Impact

Company & Customer:

Allowing unauthenticated access can lead to unauthorized data exposure and manipulation, which damages user trust and puts sensitive information at risk. Attackers can exploit this to access protected resources or perform actions intended only for authenticated users, affecting both data integrity and user experience.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Data Privacy Violations:** Exposing data without authentication may breach privacy laws and customer data protections.

- **Non-Compliance with Security Standards:** Unauthorized access violates common security standards, such as GDPR and PCI-DSS.

- **Liability for Unauthorized Actions:** If attackers misuse the system, the company may be held liable for damages.
- **Service Agreement Breach:** Exposing data meant for authenticated users may violate terms of service agreements.
- **Regulatory Fines:** Exposing sensitive information due to insufficient authentication controls may lead to penalties.

Steps for mitigation

- **Enforce Strict Authentication Checks:** Ensure JWT verification for all endpoints that require authentication.
- **Implement Access Control Policies:** Use role-based access control to restrict data and actions based on user roles.
- **Log and Monitor Unauthenticated Access Attempts:** Track and investigate unauthorized access patterns to detect misuse.
- **Use Secure JWT Handling:** Ensure JWT tokens are properly generated, verified, and secured to prevent unauthorized access.

Vulnerability no. 14

Name of Vulnerability: JWT Token Tampering

- **Affected Component: JWT token**

- **Affected URL:**

http://127.0.0.1:8888/workshop/api/shop/orders/26

Description Of Vulnerability:

A JWT (JSON Web Token) is a compact, URL-safe means of representing claims to be transferred between two parties. It's commonly used for authentication and authorization purposes in web applications. JWTs consist of three parts: the header, the payload, and the signature. The header and payload are Base64Url encoded JSON objects, and the signature is used to verify the integrity of the token.

Here in this vulnerability I can modify the JWT token information and impersonate other users.

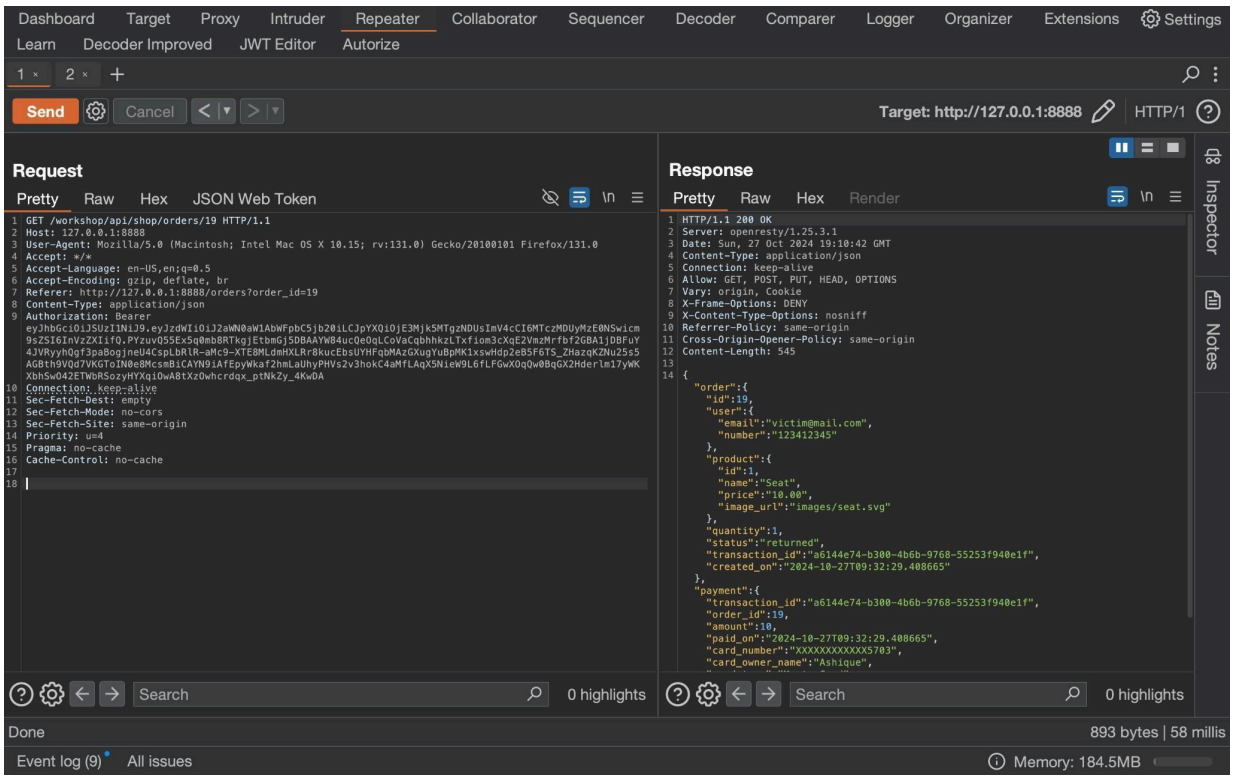
Steps to Reproduce the vulnerability:

Tools Used:

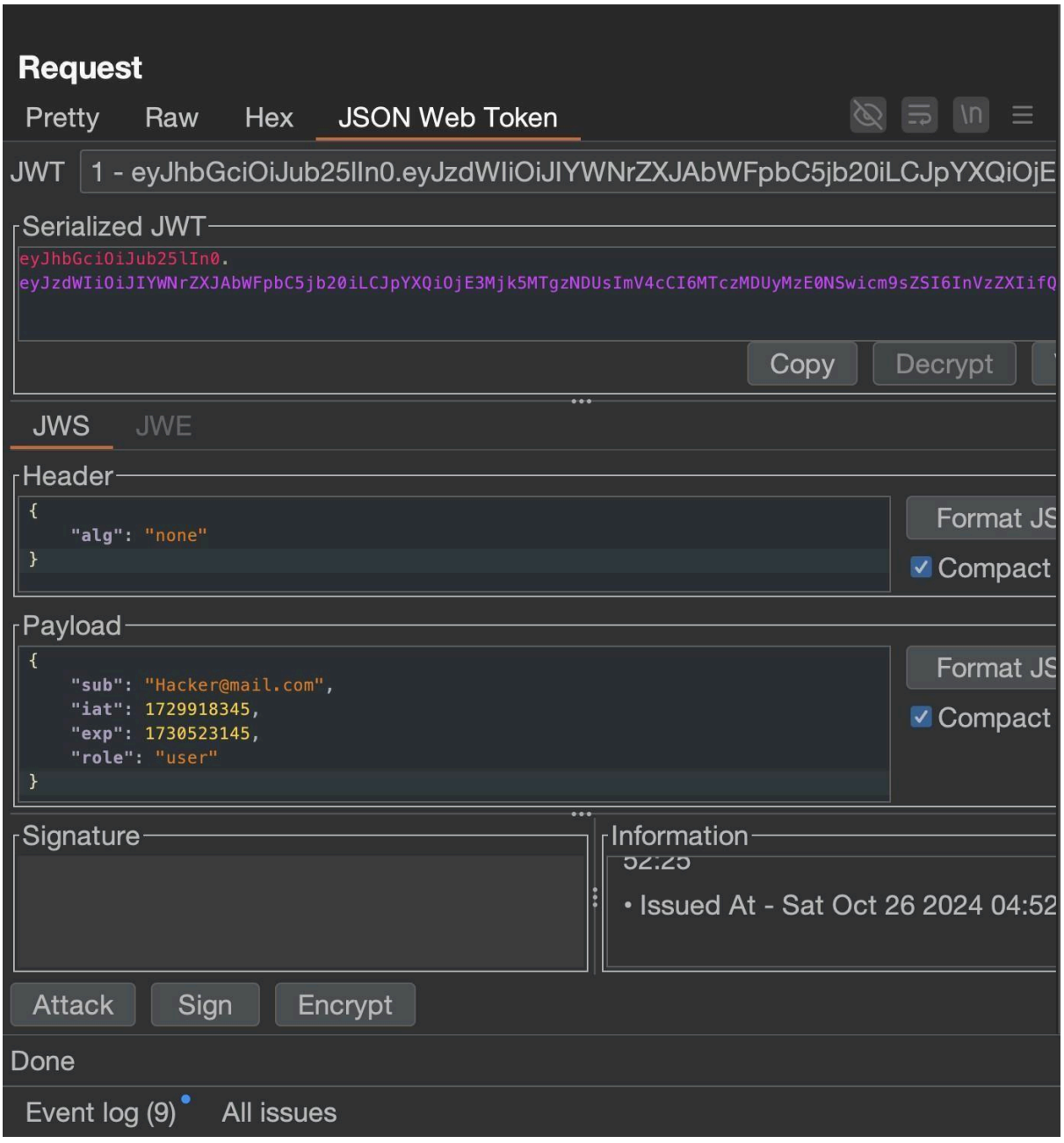
- Burp suite

1. Go to shop → click past orders → click order details → send the request to burp suite

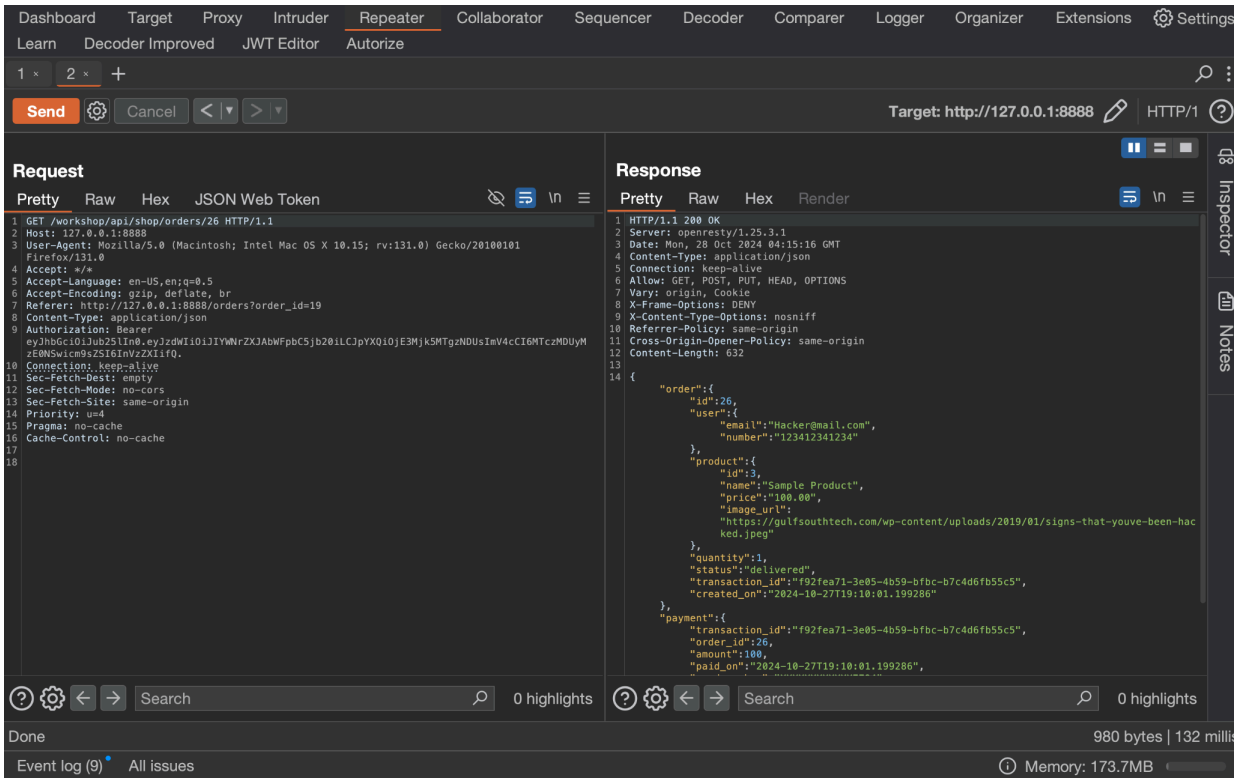
Request with bearer token (JWT):



2. Now try changing the token information like this:



3. Now send the request modified JWT:



See it returned the data of another user.

Impact

Company & Customer:

This vulnerability can allow attackers to impersonate other users, including administrators, leading to unauthorized access, data breaches, and potential financial and reputational damage.

Legal Issues:

The vulnerability may result in many legal issues some of these are possible ones:

- **Privacy Violations:** If tampered tokens expose user data, it may lead to violations of data protection laws, like GDPR or CCPA, which could result in hefty fines.
- **Breach of Confidentiality Agreements:** Unauthorized access to sensitive data can breach confidentiality clauses in contracts with clients, vendors, or partners.

- **Compliance Failures:** Many regulatory frameworks (e.g., HIPAA, PCI-DSS) require strict access control, and failing to protect tokens may lead to non-compliance penalties.
- **Liability for Damages:** If attackers misuse tampered tokens for fraud or unauthorized actions, affected users could claim damages, holding the company liable for insufficient security measures.
- **Risk of Class Action Lawsuits:** If a large number of users are affected, this could trigger class action lawsuits, especially in cases of financial or personal data exposure.

Steps for mitigation

To prevent JWT tampering:

- Use strong algorithms (like RS256) for token signing.
- Implement strict signature verification on the server.
- Protect and rotate secret keys regularly.