Examining Access Control Vulnerabilities in GraphQL

A FEELD Case Study on Data Exposure

FORTBRIDGE

Whoami?

Bogdan Tiron

> 10 years of experience in security

Senior Pentester at FORTBRIDGE

Accreditations: OSCP, CREST CRT, DevSecOps, GCP Security, GCP Architecture

Past History: Lloyds Bank, GFK, JPMorgan Bank, bet365

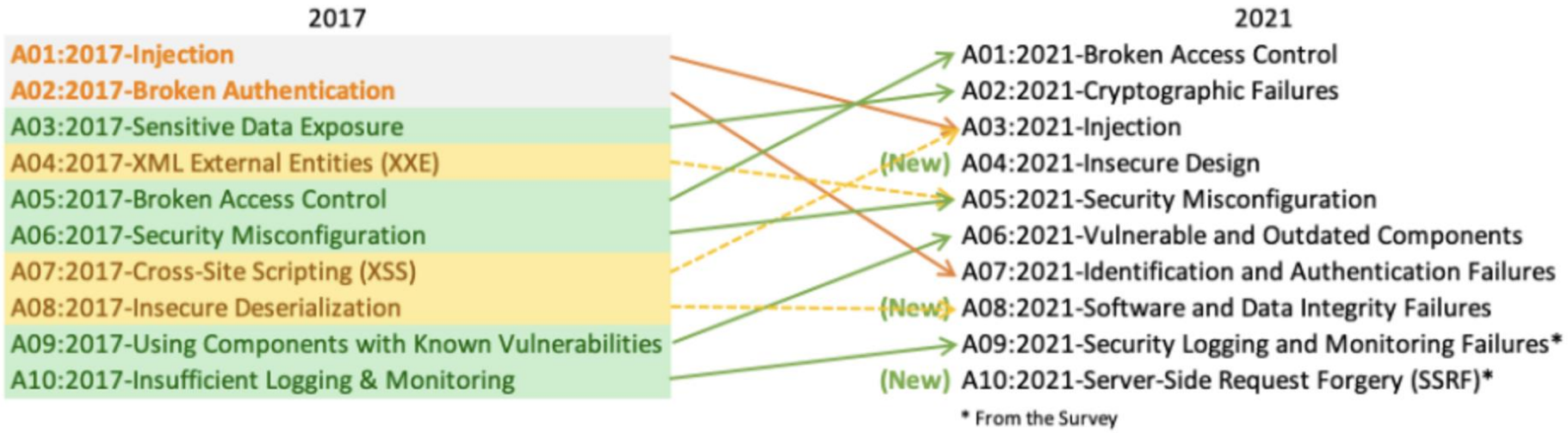
WHAT IS THIS TALK ABOUT?

This is about The Importance of Access Controls

OWASP Top 10 - #1 Security Risk

OWASP Top 10 API Security Risks – 2023

Risk	Description
API1:2023 - Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface of Object Level Access Control issues. Object level authorization checks should be considered in every function that accesses a data source using an ID from the user.

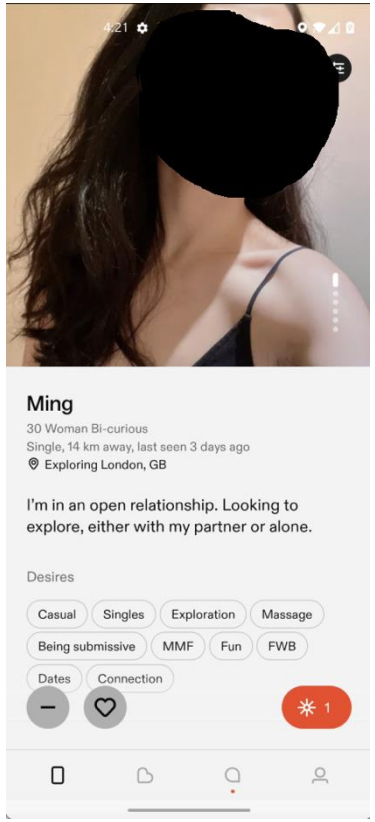


- APIs: #1 Broken Object Level Authorization Category
- Web Apps: #1 Broken Access Controls Category

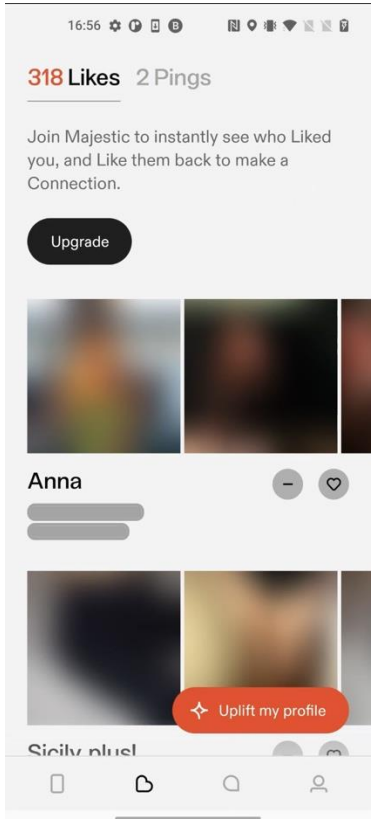
About Feeld

- a dating mobile app, like Tinder/Bumble/etc
- you can filter by distance, by age, by gender (>10), couples, and by location.
- for **premium** users, you can also search by the type of kink, threeways/group scenarios, or the type of relationship you are interested in.
- >1 Million Downloads (Android Play Store)

FEELD Menus

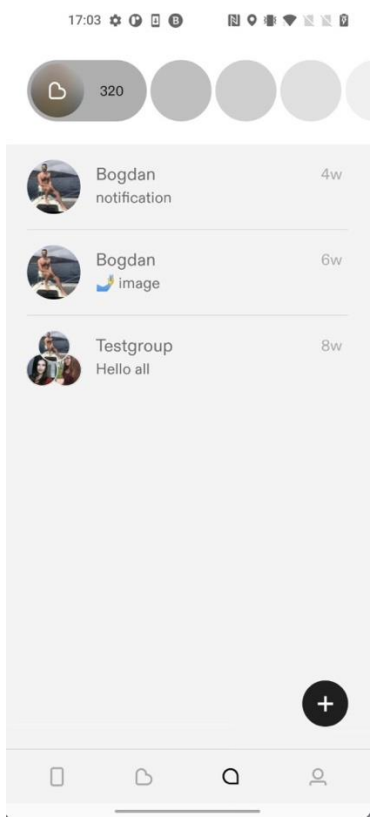


'Discover Profiles' Menu

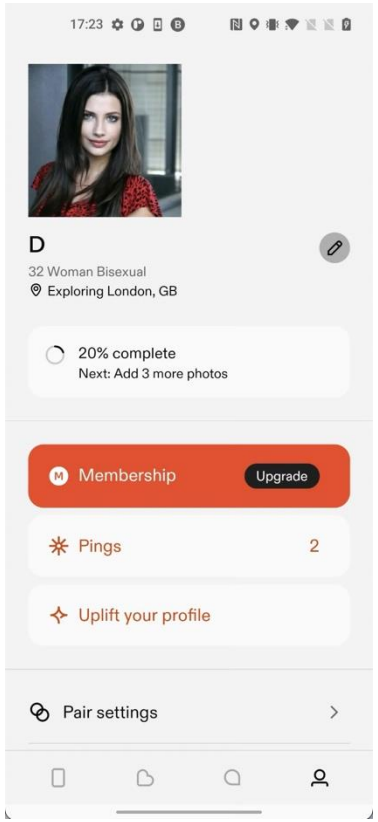


'Who liked you' Menu

'basic' users cannot view/interact with it



'Messages' Menu



'My Profile' Menu

FEELD Case Study - Vulnerabilities

1.Disclosure of profile information to non-premium users

2.Read other people's messages

3.Unauthenticated access to other people's attachments (photos & videos) from their chats

4.Delete, recover and edit other people's messages

5.Update someone else's profile information

6.Get a 'Like' from any user profile

7.Send messages in other people's chat

8.View other people's matches

#3 Broken Object Property Level Authorization Category

#1 Broken Object Level Authorization Category

Vulnerability #1 - Disclosure of profile information to non-premium users

#3 Broken Object Property Level Authorization Category

Details: The ‘basic’ user will no longer need to pay for a ‘premium’ subscription to get a premium benefit.

- As a **basic** user, in the ‘Likes’ menu, you see who liked your profile, but you only get limited information, such as:
  - the name and
  - the blurred photos of the ‘like’ sender,
- As a **premium** user who gets all the information available about the sender.

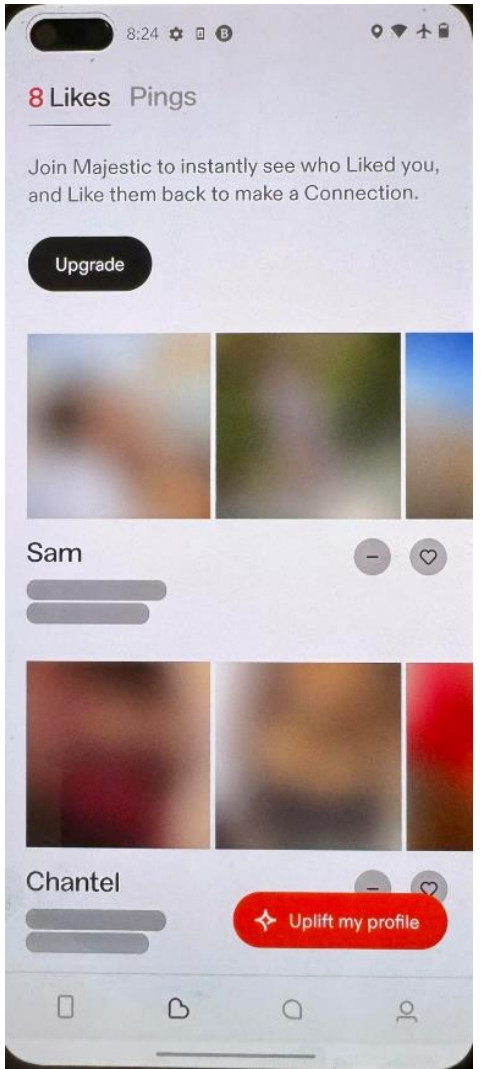
However, if you use a proxy tool such as Burp to intercept the request and response, you will find in the response all the information available about the ‘like’ senders, just like a premium user.

Vulnerability #1 - Disclosure of profile information to non-premium users

#1 Broken Object Level Authorization Category

Reproduction steps:

1. Go to the ‘Likes’ menu to see who liked or pinged us, as seen on the right. But beside their names and their blurred photos, we do not have any other information.

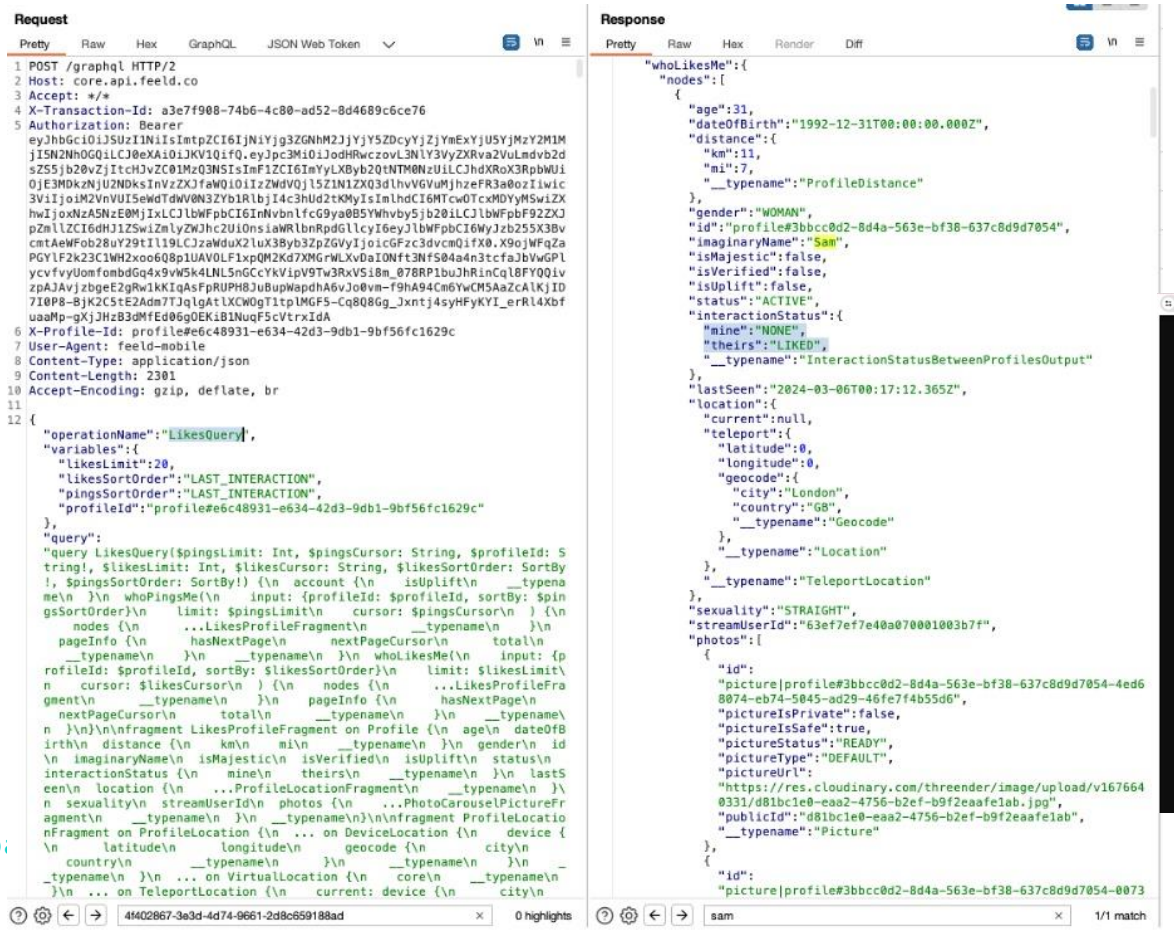


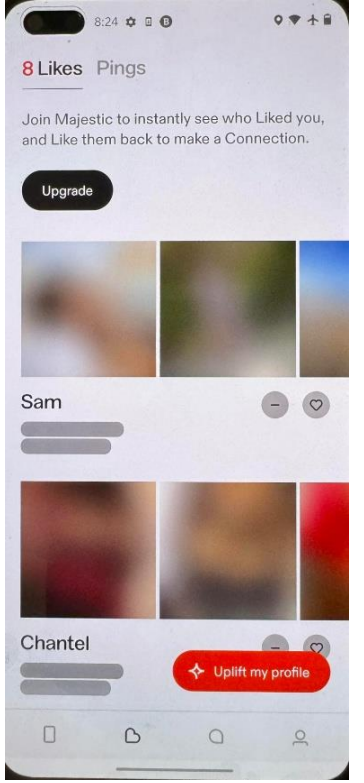
Vulnerability #1 - Disclosure of profile information to non-premium users

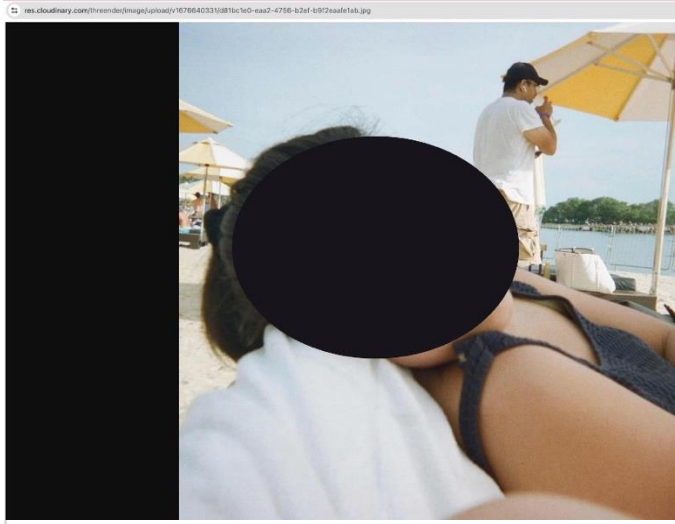
Reproduction steps:

2. However, if we intercept the request in Burp and check the response, as seen below, we will see that we have all the information about the user (age, distance, all their profile photos, streamUserId), including **unauthenticated** access to their profile photos stored on res.cloudinary.com.

In addition, using the '**streamUserId**' value found in the response we can exploit the next vulnerability 'Read other people's messages' and read Sam's messages.







Vulnerability #2 - Read other people's messages

#1 Broken Object Level Authorization Category

Details: We can read other people's messages in the chat.

In order to do that, we will need to get our victim's 'streamUserId' value, which is disclosed in different API requests.

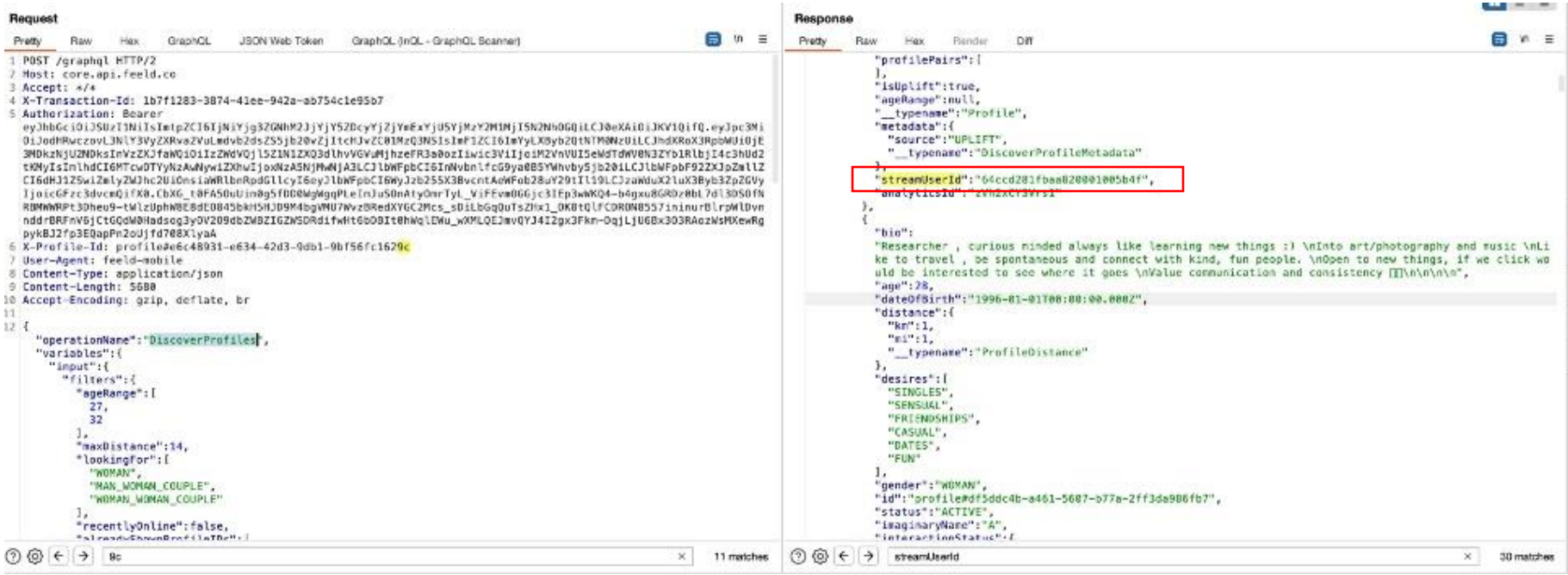
Vulnerability #2 – Read other people's messages

Reproduction steps:

1.Go to the 'Discover profiles' menu.

2.Intercept the /graphql request with operationName: 'DiscoverProfiles'.

Get a 'streamUserId parameter value of the target user from the response, as seen on the right:



'StreamUserId' is: 64ccd281fbaa820001005b4f , and belongs to user 'Chlo'.

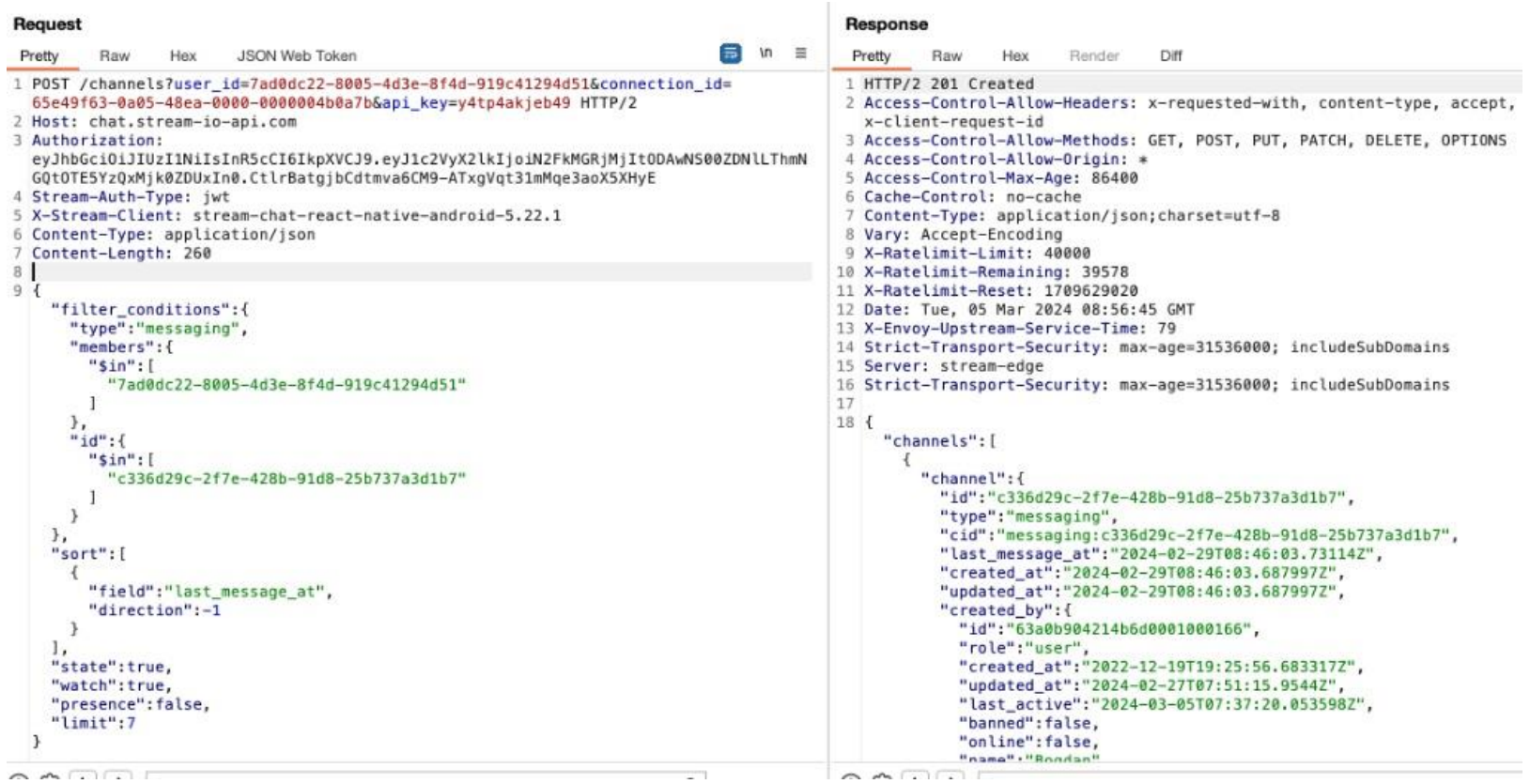
Vulnerability #2 – Read other people's messages

Reproduction steps:

3. Now go to the 'Message' menu, and intercept the request to the endpoint:

https://chat.stream-io-api.com/channels?user_id=&connection_id=&api_key=y4tp4akjeb49

,such as the one below:



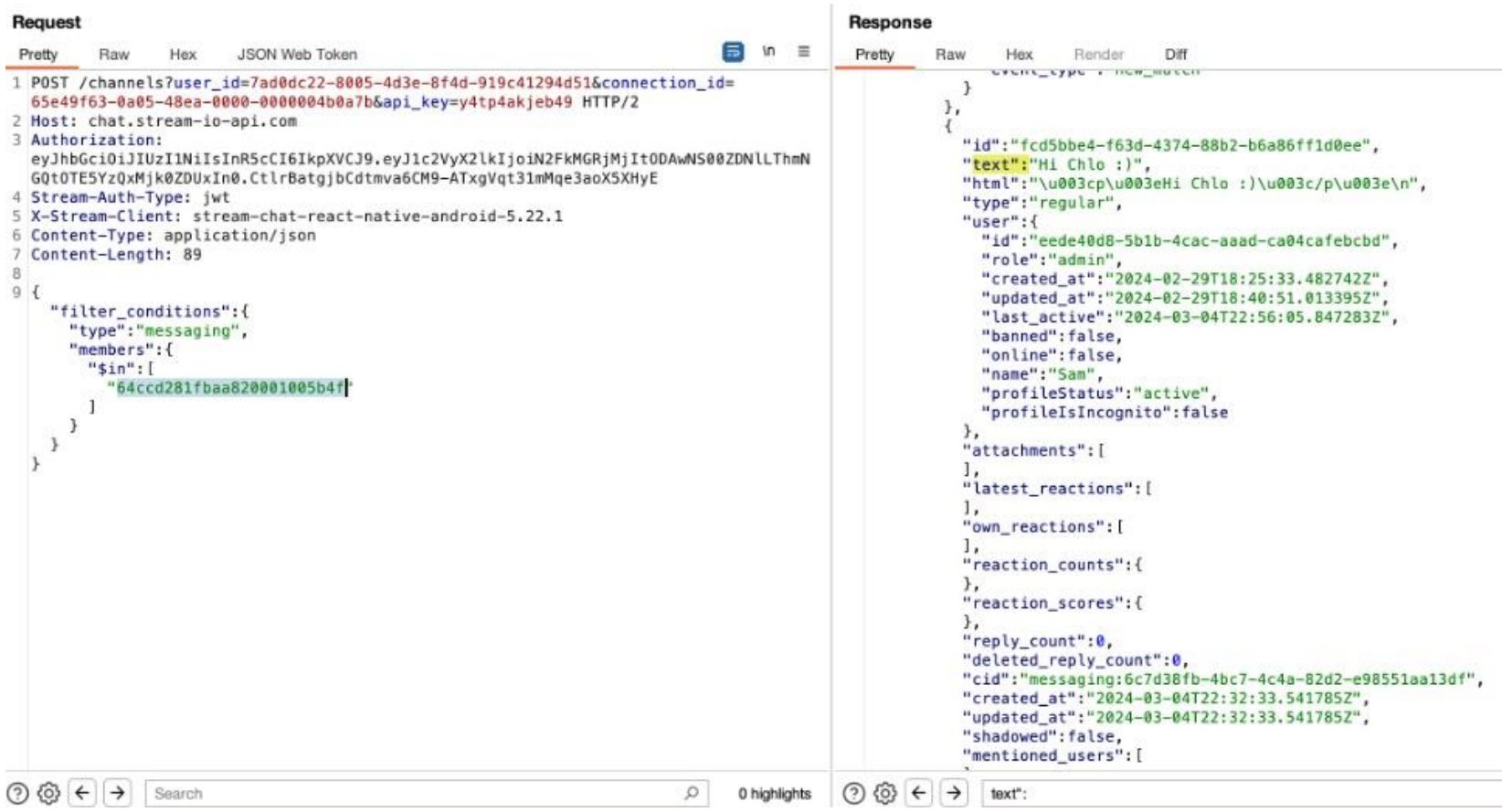
Vulnerability #2 – Read other people's messages

Reproduction steps:

4. Remove all the request parameters except:

'member':{'in':["<value>"] ,

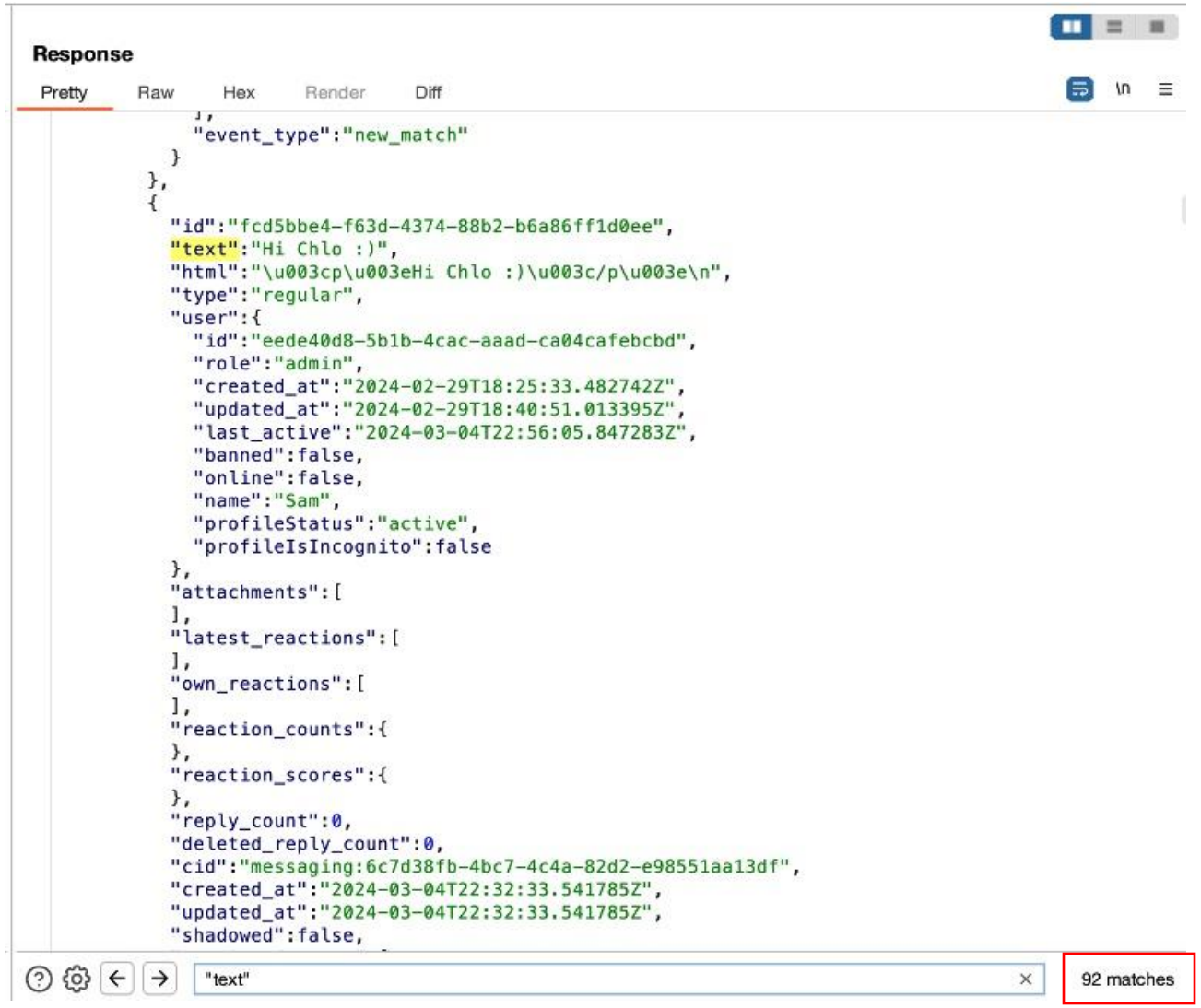
and add the victim's 'streamUserId' as <value> , as seen below:



Vulnerability #2 – Read other people's messages

Reproduction steps:

5.If we search in the response by "text" we can see the total number of messages to and from our victim 'Chloe:



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

#1 Broken Object Level Authorization Category

Details: We can build upon the previous issue: "Read other people's messages".

To do that, we will need to get our victim's 'streamUserId' value, which is disclosed in different API requests.

There are 2 types of attachments:

1. Photos
   - Replay-able
   - Time-limited (5-15 seconds – after which becomes unavailable)
2. Videos
   - Replay-able
   - Play-once only

As an attacker, we can access all of the following unauthenticated:

- Photos replay-able
- Photos time-limited
- Videos replay-able
- Videos play-once

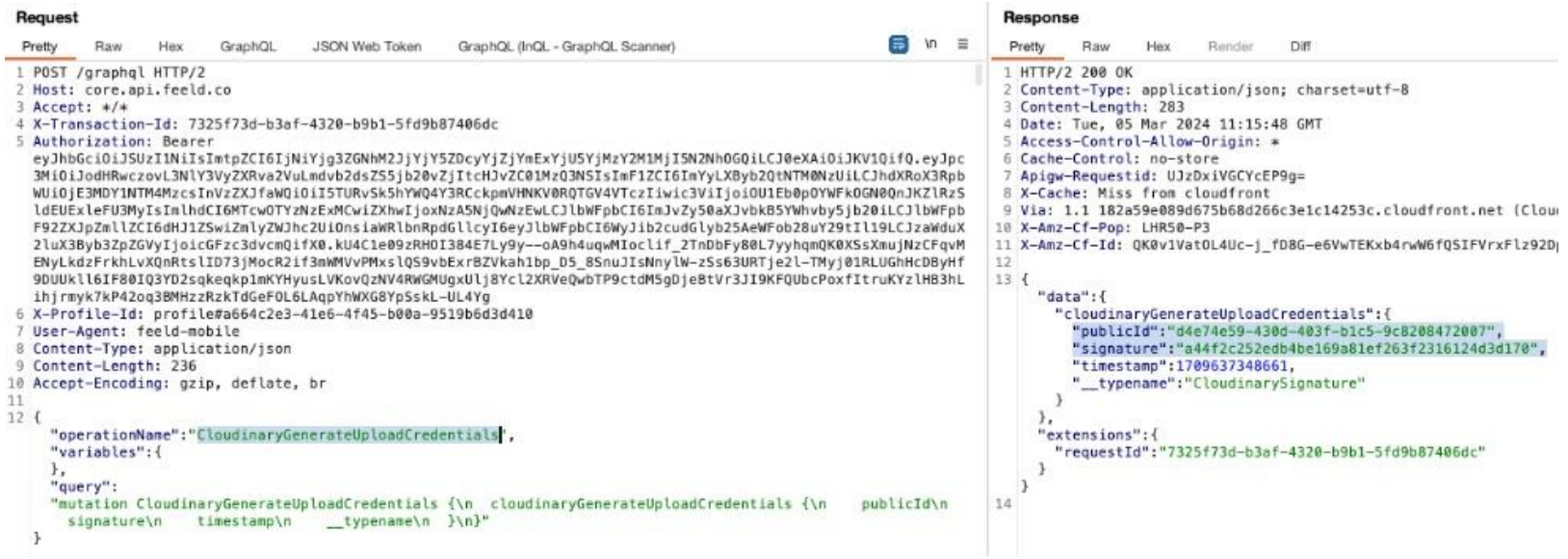
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 1: Uploading replay-able photos

1.Let's upload in our chat, a normal replay-able photo.

So, the first request is 'Generate Upload Credentials' for uploading on 'api.cloudinary.com':

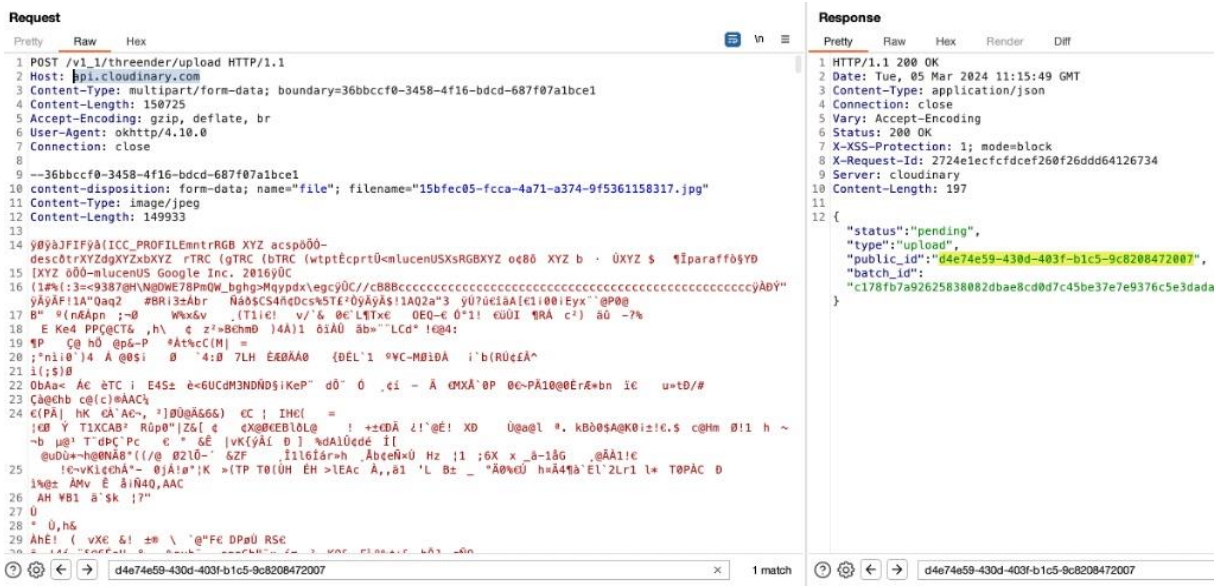


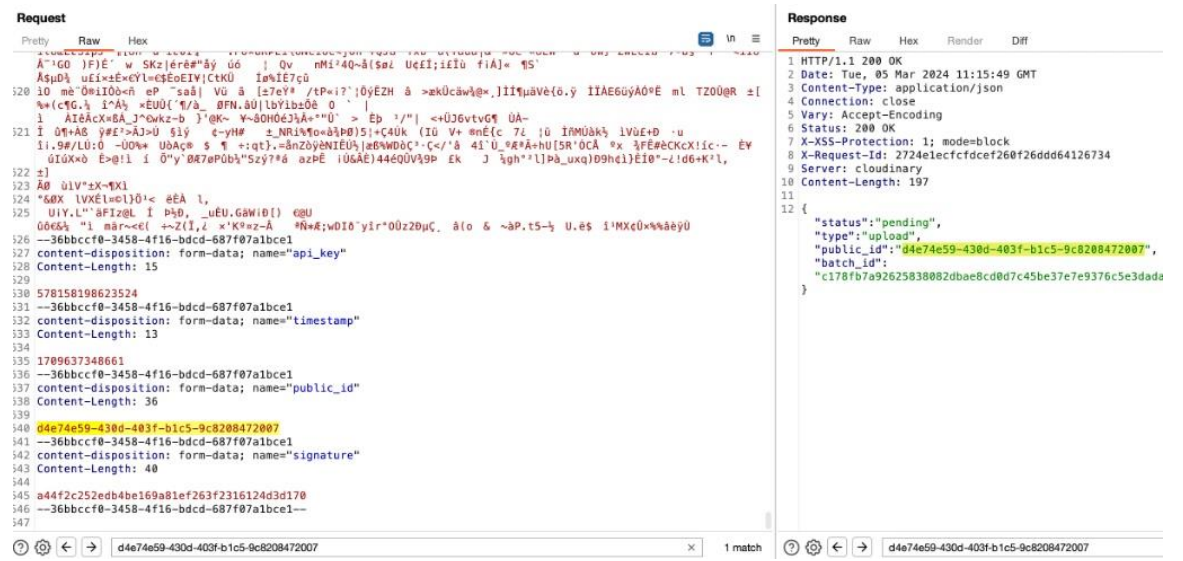
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 1: Uploading replay-able photos

2. Then we send a photo upload request to api.cloudinary.com using the above generated 'publicId' and 'signature' values, plus an api_key and timestamp parameters:





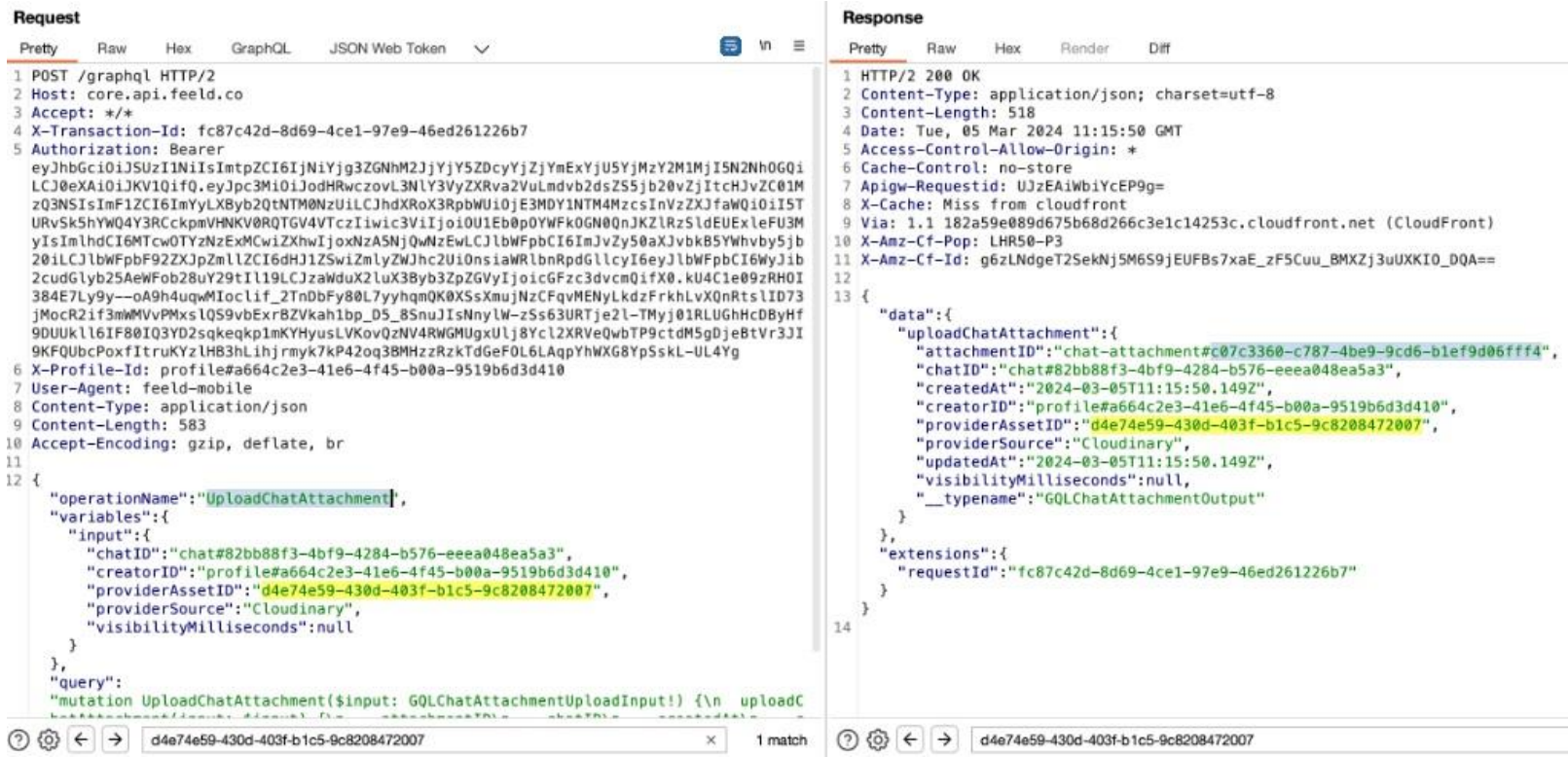
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 1: Uploading replay-able photos

3. Next request done is: 'UploadChatAttachment' which gets the above unique public_id of the image from api.cloudinary.com and is passed to core.api.feeld.co, as seen below.

I suspect this request is to copy the photo from api.cloudinary.com to core.api.feeld.co.



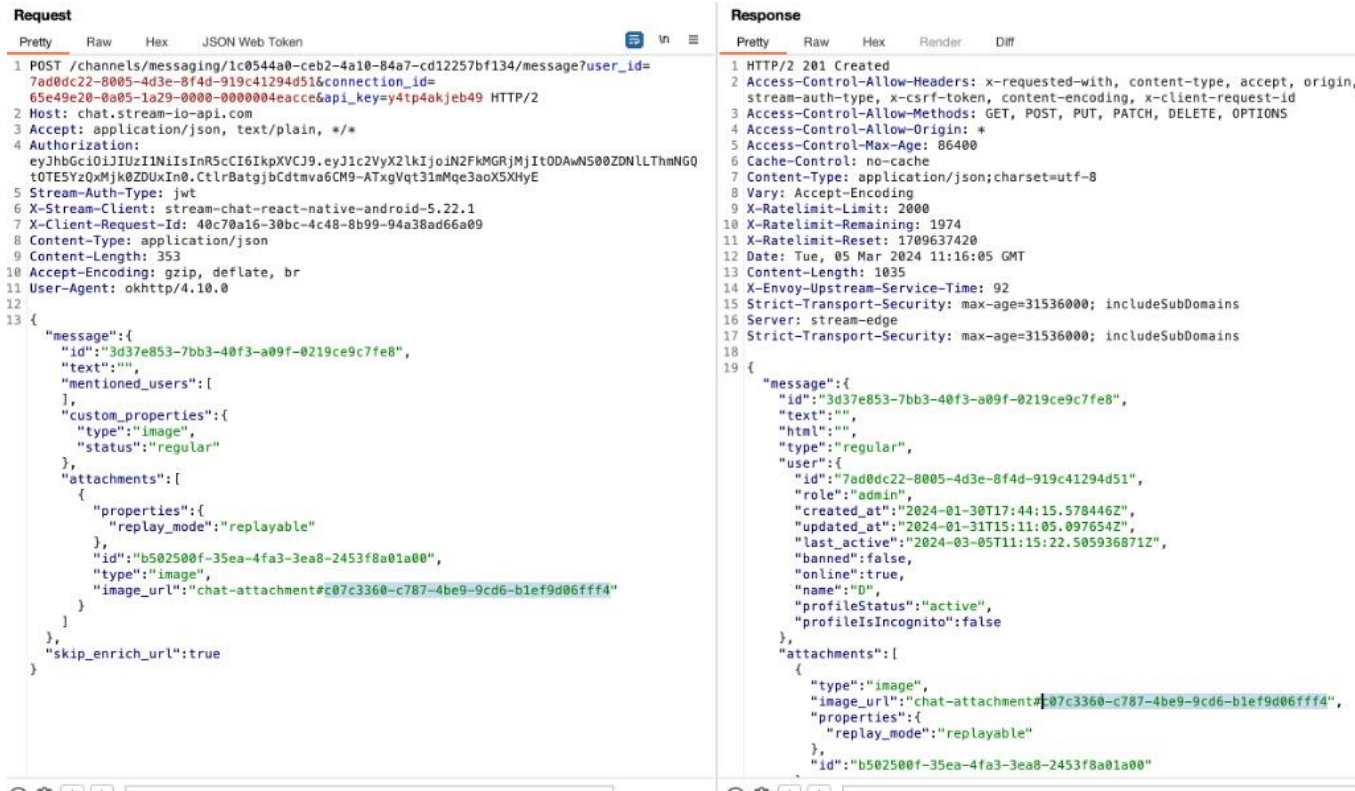
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 1: Uploading replay-able photos

4. A unique 'attachmentID' parameter will be returned above in the response.

This 'attachmentID' will be used and passed in the chat, as seen below:



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

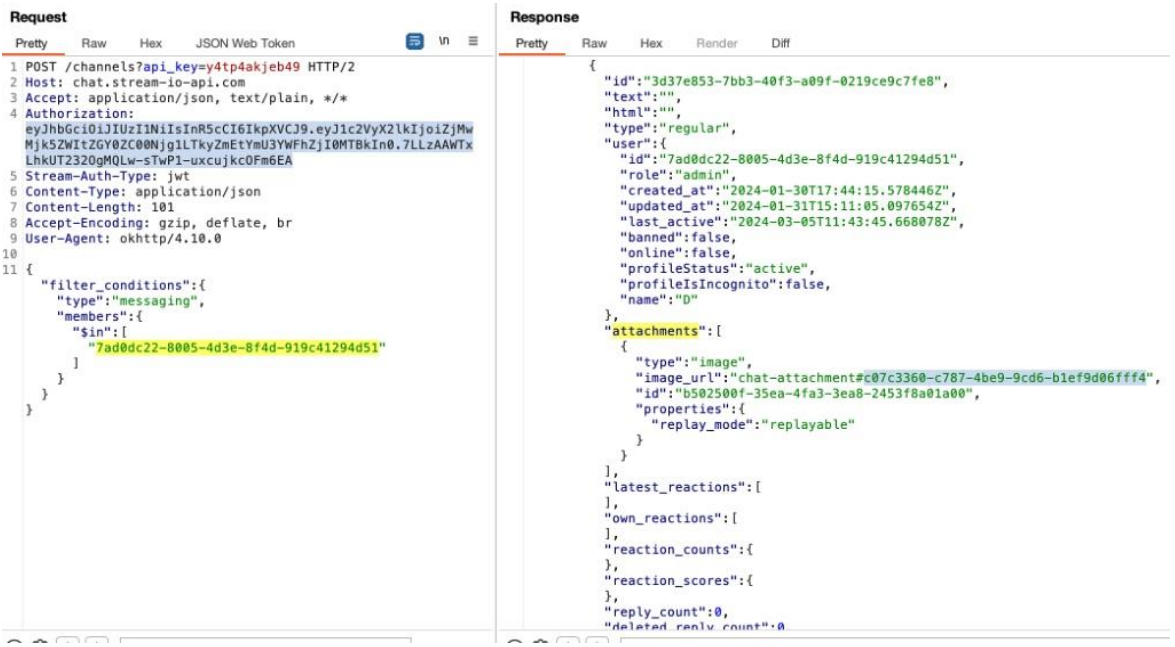
Reproduction steps:

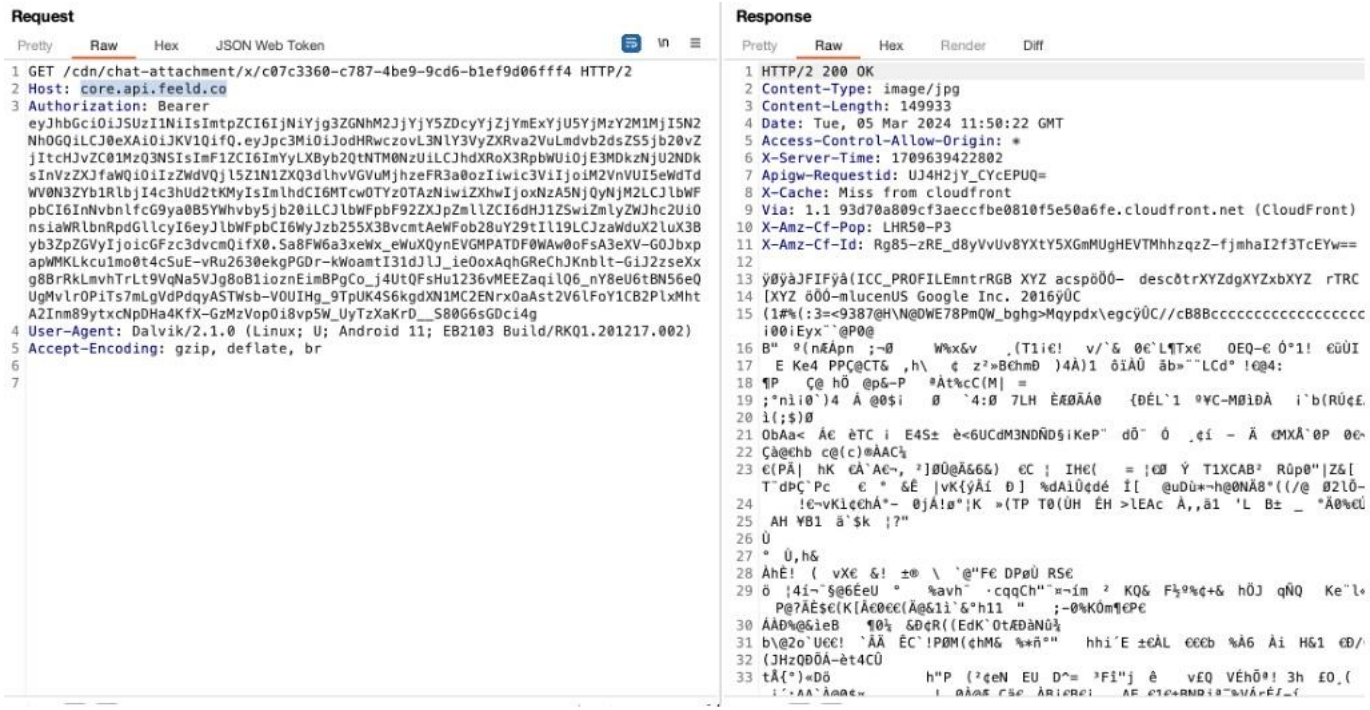
Instance 1: Uploading replay-able photos

5. Now to get the photo authenticated, as any other user, we make the following request, using the above attachmentID:

https://core.api.feeld.co/cdn/chat-attachment/x/**c07c3360-c787-4be9-9cd6-b1ef9d06fff4**

Note: In the above request path, initially instead of 'x' it was the 'ProfileId' guid value of the sender or receiver of the photo, but deleting it works fine, so I just left an 'x' for an easier read.





Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 1: Uploading replay-able photos

6.Now, to get the same photo but from cloudinary.com, **unauthenticated**, prepend /v1/ to the above request, as seen below, and you will get the 'url' pointing to the original photo:

https://res.cloudinary.com/threender/image/upload/s--QQjZiJxc--/d4e74e59-430d-403f-b1c5-9c8208472007

Request

Pretty Raw Hex JSON Web Token

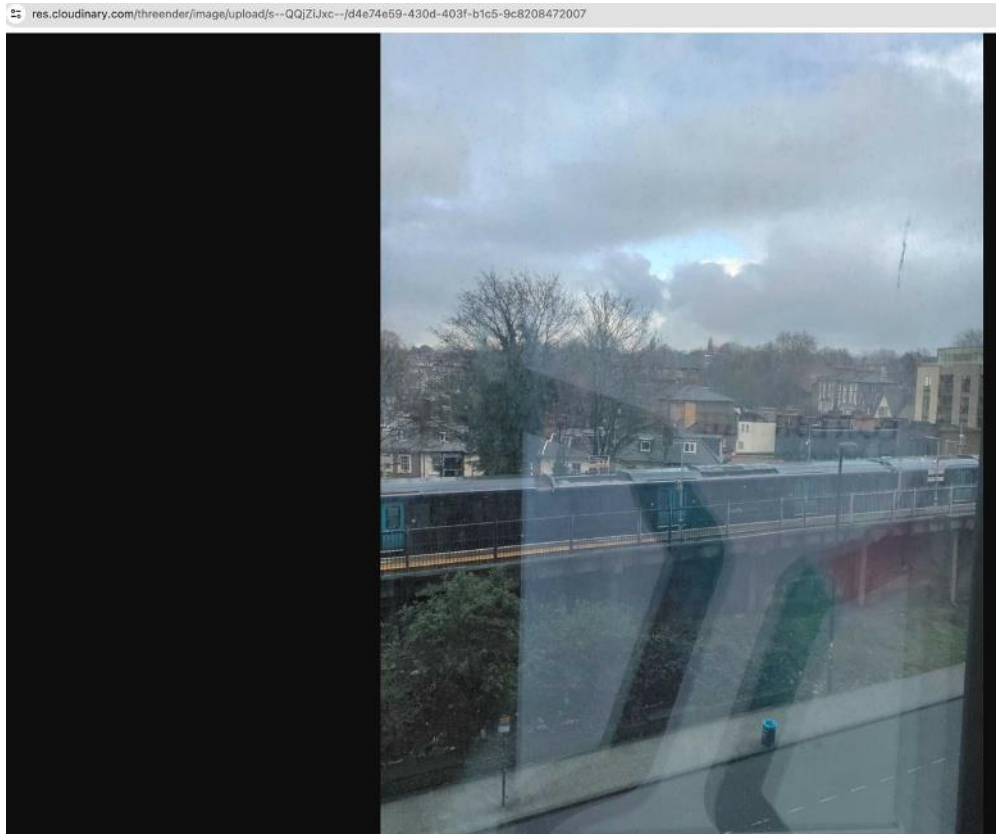
```
GET /v1/cdn/chat-attachment/x/c07c3360-c787-4be9-9cd6-b1ef9d06fff4 HTTP/2
Host: core.api.feeld.co
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjNiYjg3ZGNhM2JjYjY5ZDcyYjZjYmExYjU5YjMzY2M1MjI5N2
NhOGQiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vZ
jItcHJvZC01MzQ3NSIsImF1ZCI6ImYyLXByb2QtNTM0NzUiLCJhdXRoX3RpbWUiOjE3MDkzNjU2NDk
sInVzZXJfaWQiOiIzZWdVQjl5Z1N1ZXQ3dlhvVGVuMjhzeFR3a0ozIiwic3ViIjoiM2VnVUI5eWdTd
WV0N3ZYb1Rlbjl4c3hUd2tKMyIsImlhdCI6MTcwOTYzOTAzNiwiZXhwIjoxNzA5NjQyNjM2LCJlbWF
pbCI6InNvbnlfcG9ya0B5YWhvby5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmlyZWJhc2UiO
nsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJzb255X3BvcmtAeWFob28uY29tIl19LCJzaWduX2luX3B
yb3ZpZGVyIjoicGFzc3dvcmQifX0.Sa8FW6a3xeWx_eWuXQynEVGMPATDF0WAw0oFsA3eXV-GOJbxp
apWMKLkcu1mo0t4cSuE-vRu2630ekgPGDr-kWoamtI31dJlJ_ieOoxAqhGReChJKnblt-GiJ2zseXx
g8BrRkLmvhTrLt9VqNa5VJg8oB1ioznEimBPgCo_j4UtQFsHu1236vMEEZaqilQ6_nY8eU6t8N56eQ
UgMvlrOPiTs7mLgVdPdqyASTWsb-VOUIHg_9TpUK4S6kgdXN1MC2ENrx0aAst2V6lFoY1C82PlxMht
A2Inm89ytxcNpDHa4KfX-GzMzVopOi8vp5W_UyTzXaKrD__S80G6sGDci4g
User-Agent: Dalvik/2.1.0 (Linux; U; Android 11; EB2103 Build/RKQ1.201217.002)
Accept-Encoding: gzip, deflate, br
```

Response

Pretty Raw Hex Render Diff

```
HTTP/2 200 OK
Content-Type: text/plain; charset=utf-8
Content-Length: 100
Date: Tue, 05 Mar 2024 11:51:32 GMT
Access-Control-Allow-Origin: *
X-Server-Time: 1709639492282
Apigw-Requestid: UJ4StiFSiYcEMjQ=
X-Cache: Miss from cloudfront
Via: 1.1 ed5bf73eea0876436de4cbcd6f6945e4.cloudfront.net (CloudFront)
X-Amz-Cf-Pop: LHR50-P3
X-Amz-Cf-Id: UehlMT3FvozcgOwJxcOeCjoZPRyTAscz6NmBPqWpyxVHL9RciEKcDg==

https://res.cloudinary.com/threender/image/upload/s--QQjZiJxc--/d4e74e59-430d-403f-b1c5-9c8208472007
```



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 2: Uploading time-limited photos

Note: There are 2 main differences from the previous process for replay-able photos.

1. *when uploading 'time-limited' photos, we pass an extra parameter 'visibilityMilliseconds:15000'.*
2. *for accessing the photo, we use the 'profileId' GUID value of the victim that uploaded the photo, rather than the 'x' value used in the path for replay-able photos.*

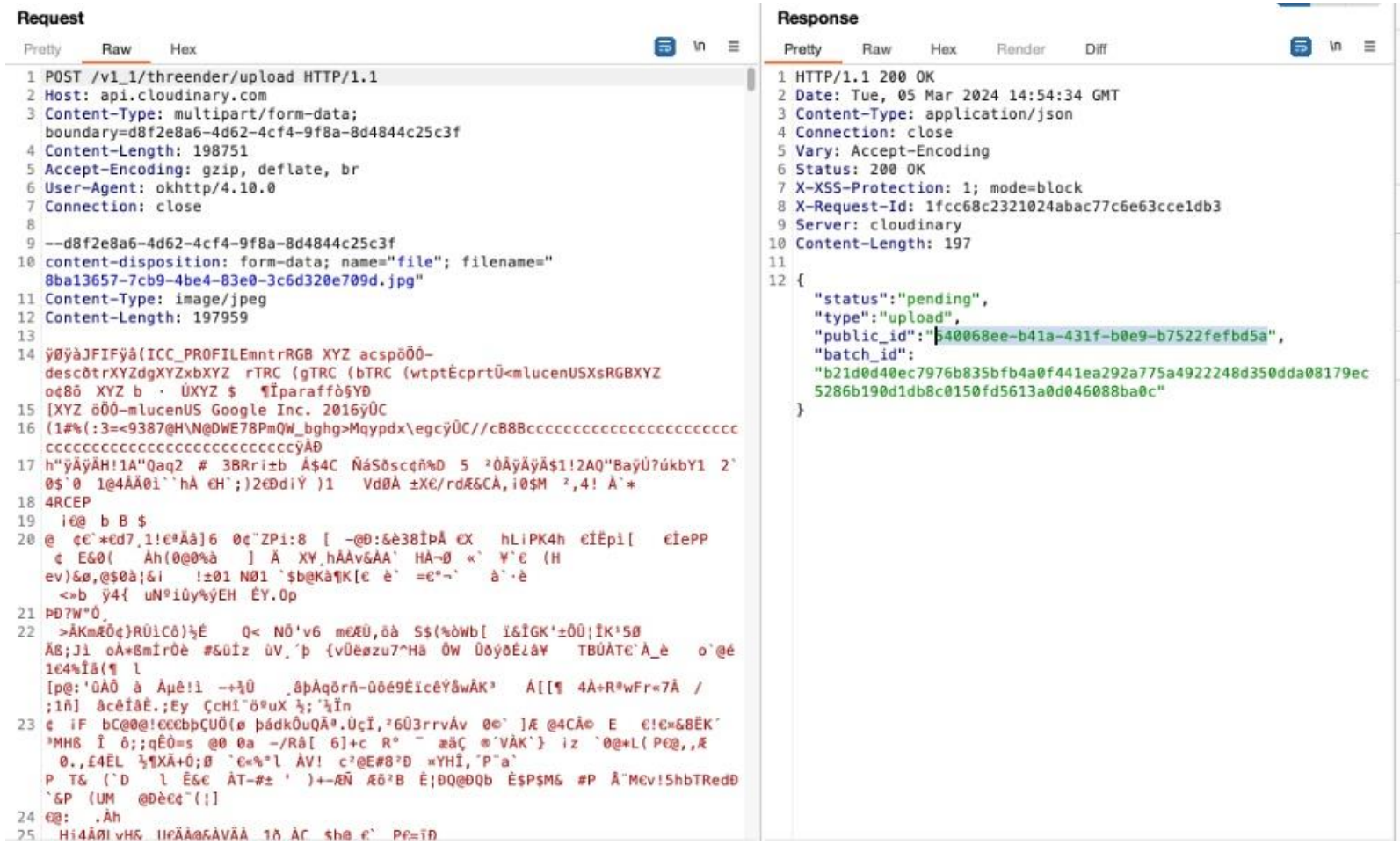
   If their chat counterpart uploaded the 'time-limited' photo, we need to return to the 'Discover profiles' menu to locate their <profileId> GUID value, which is mandatory for accessing these photos.

Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 2: Uploading time-limited photos

1. A request will be made to api.cloudinary.com to upload the photo:



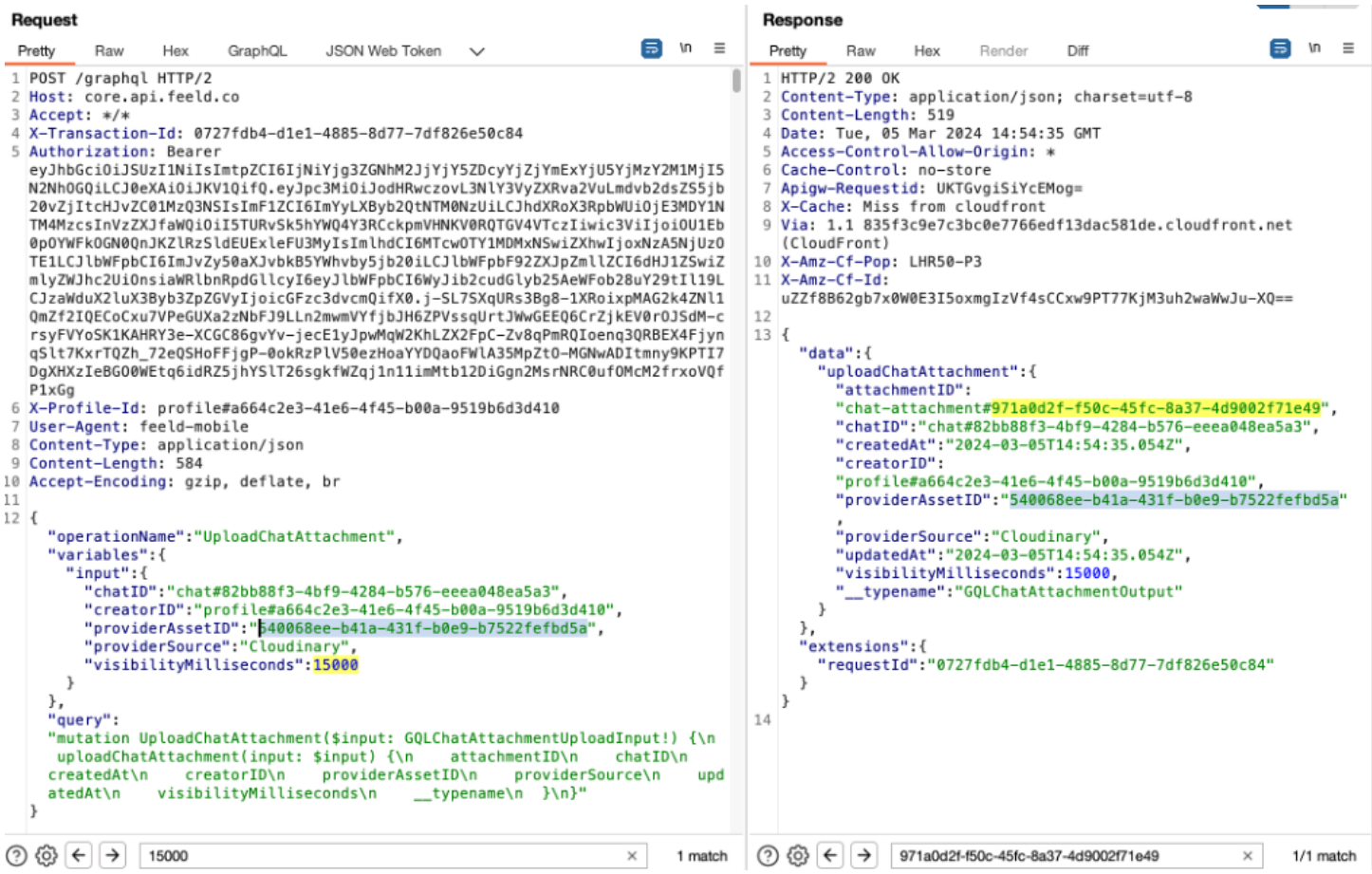
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 2: Uploading time-limited photos

2. Copying the photo from:

cloudinary.com to core.api.feeld.co:

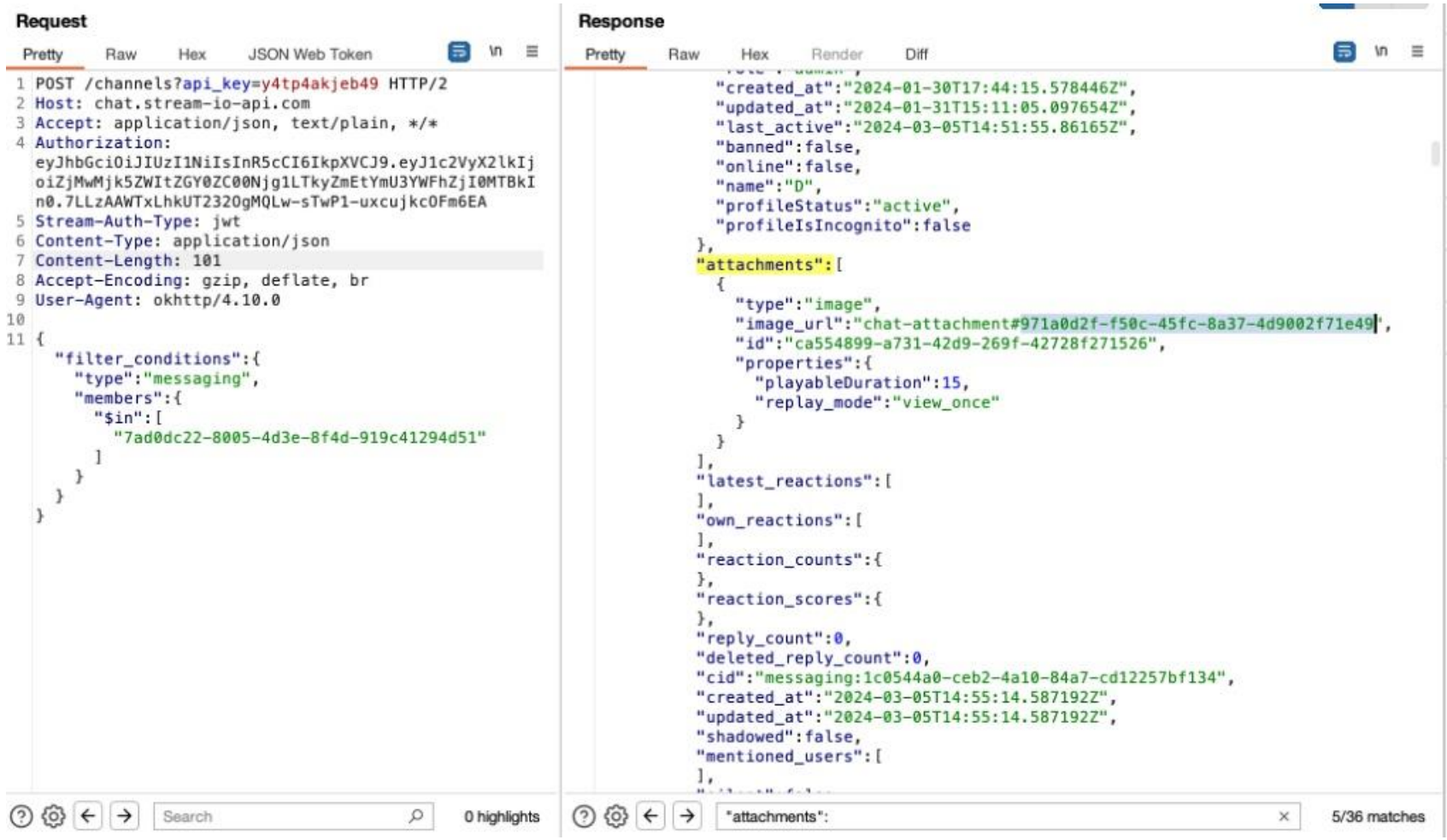


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 2: Uploading time-limited photos

3.Then, if we read again the chat using the previous vulnerability 'Read other people's messages', we can find the attachmentId to use in order to get the photo:



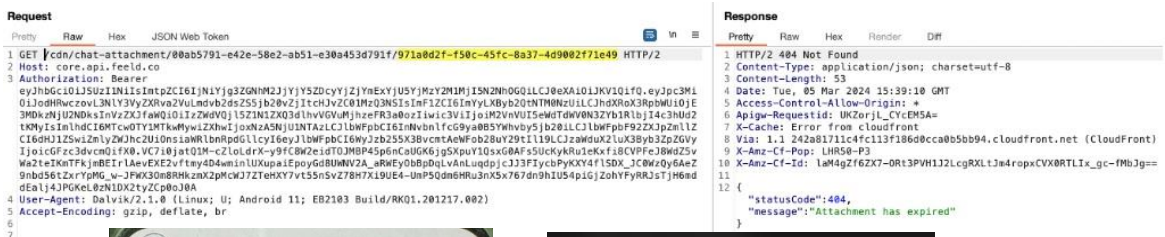
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

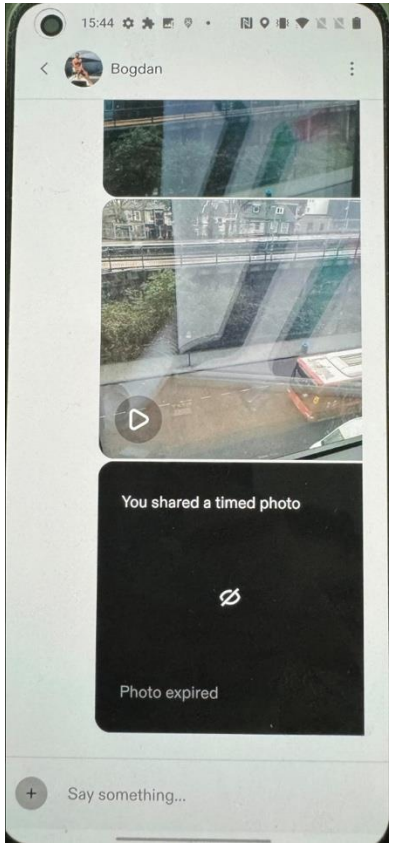
Reproduction steps:

Instance 2: Uploading time-limited photos

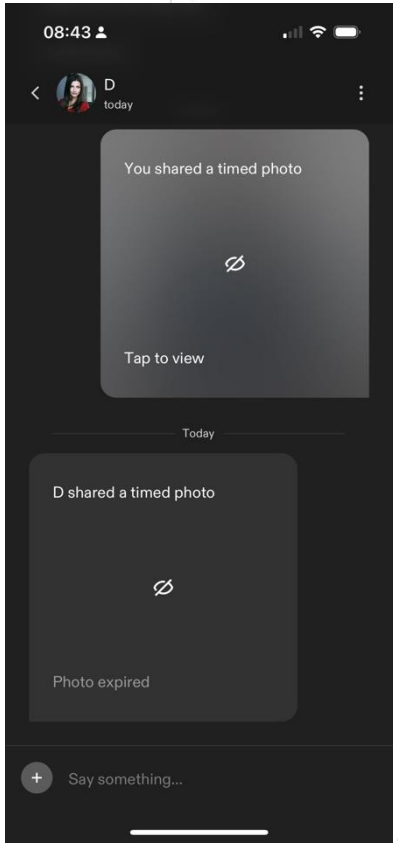
4.To retrieve the image, we will need the 'profileId' guid of our victim that uploaded the photo, which we already have from the 'Discover Profile' menu when we have chosen this target victim. Thus, the 2 urls to get the photo authenticated are:

- https://core.api.feeld.co/cdn/chat-attachment/**<receiver's_profileId>**/971a0d2f-f50c-45fc-8a37-4d9002f71e49 . However, 5-15 seconds after accessing this endpoint, the photo at this endpoint will be deleted . You must access it before the receiver.
- https://core.api.feeld.co/cdn/chat-attachment/**<victim's_profileId>**/971a0d2f-f50c-45fc-8a37-4d9002f71e49 . This will always return the photo to authenticated users.





The view in the Android app after the sender accessed his time-limited photo: 'Photo expired', and is not shown anymore in the app.



The view in the iOS app after the sender accessed his time-limited photo at the top: 'Tap to view', and is still shown in the app.

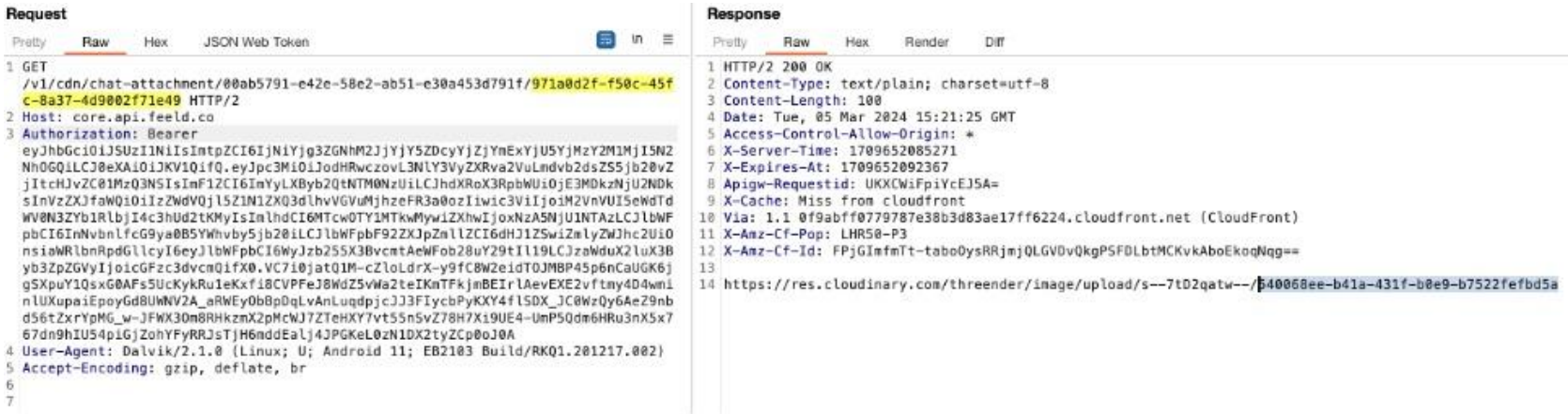
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 2: Uploading time-limited photos

5. We can use the following endpoint:

https://core.api.feeld.co/**v1**/cdn/chat-attachment/<victim's_profileId>/971a0d2f-f50c-45fc-8a37-4d9002f71e49 which will return a url with the photo stored on res.cloudinary.com .



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

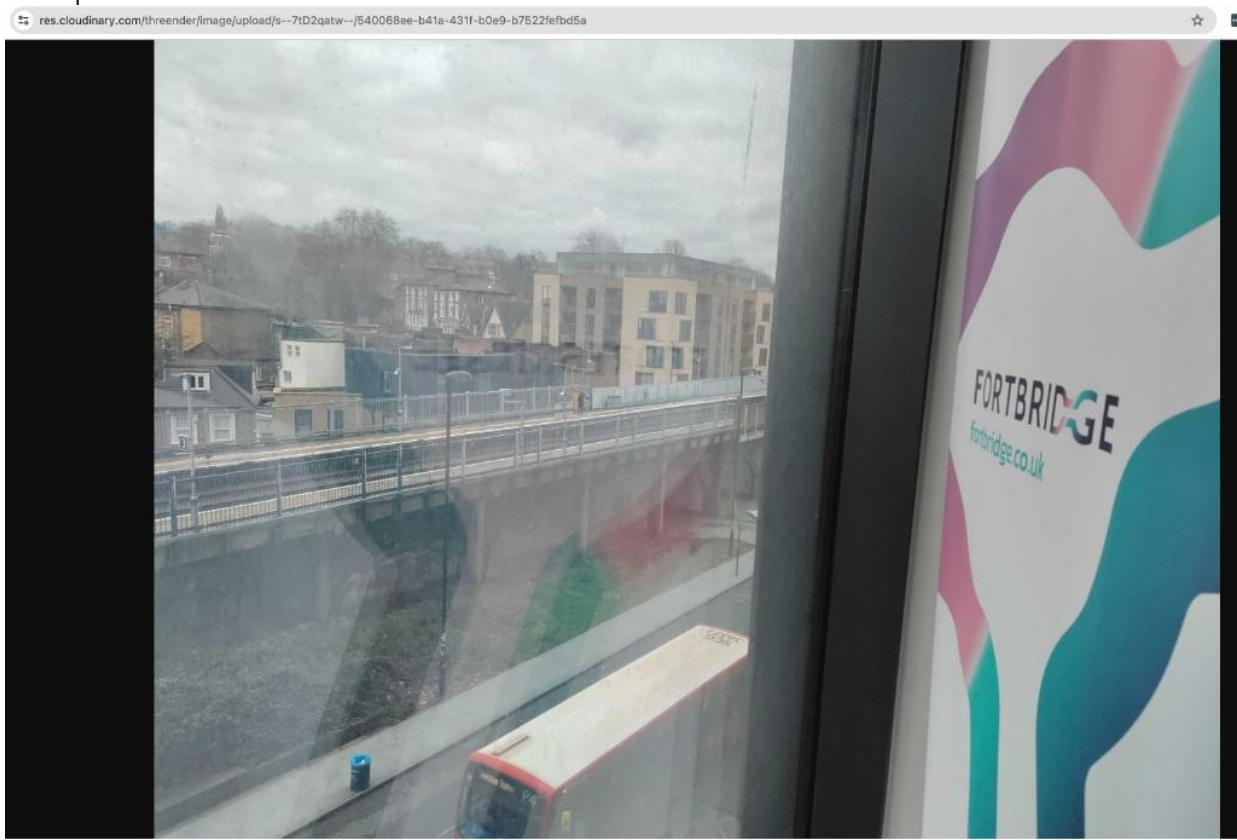
Reproduction steps:

Instance 2: Uploading time-limited photos

6. The returned url for **unauthenticated** access is:

https://res.cloudinary.com/threender/image/upload/s–7tD2qatw–/540068ee-b41a-431f-b0e9-b7522fefbd5a

The only thing random in the above url, in case you want to brute-force it, are the 8 characters '7tD2qatw'.



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

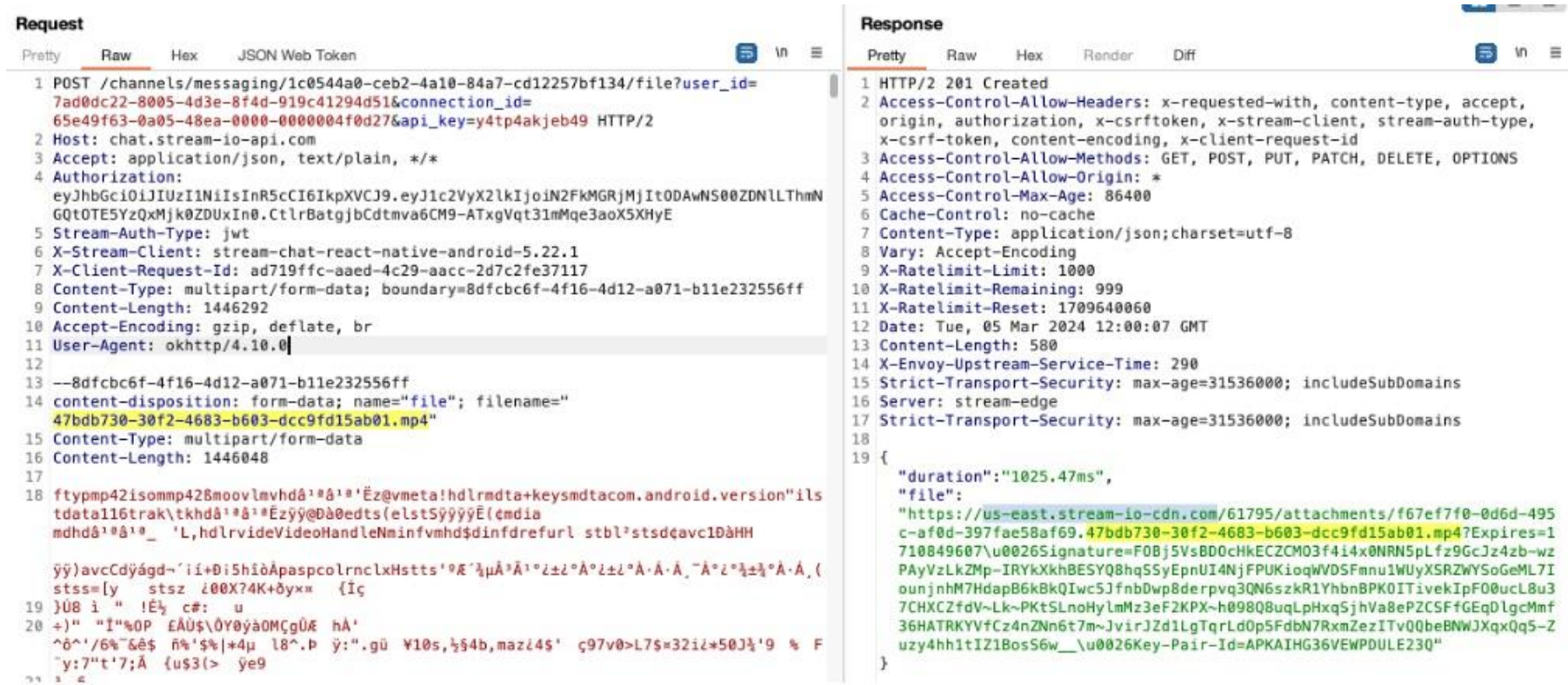
Instance 3: Uploading normal video

1.Pick a chat, select 'upload video' option, record a video and submit it in the chat.

The below requests will be made.

The video will be uploaded to:

us-east.stream-io-cdn.com:

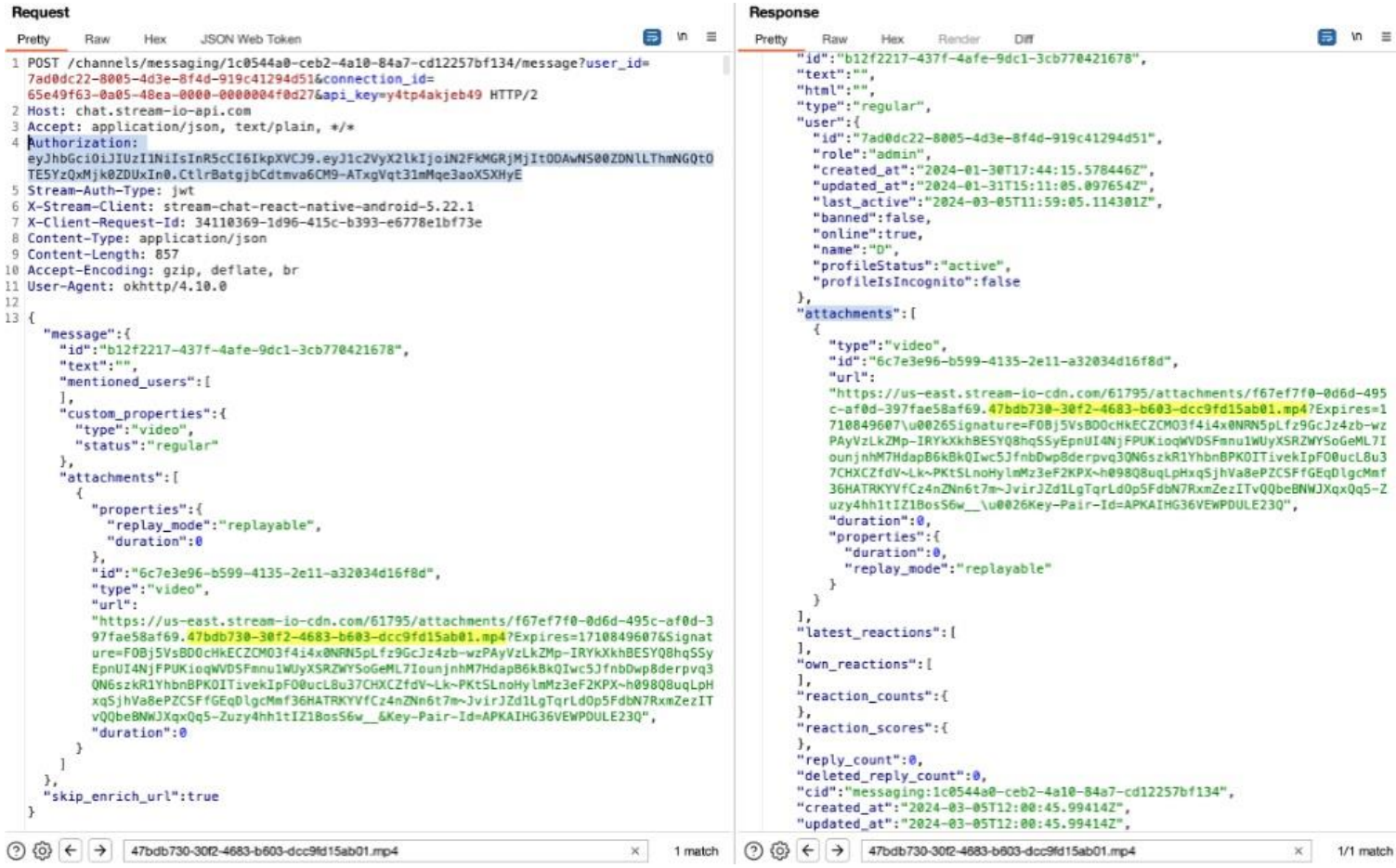


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 3: Uploading normal video

2.The url from the response will be passed in the chat, as seen below:



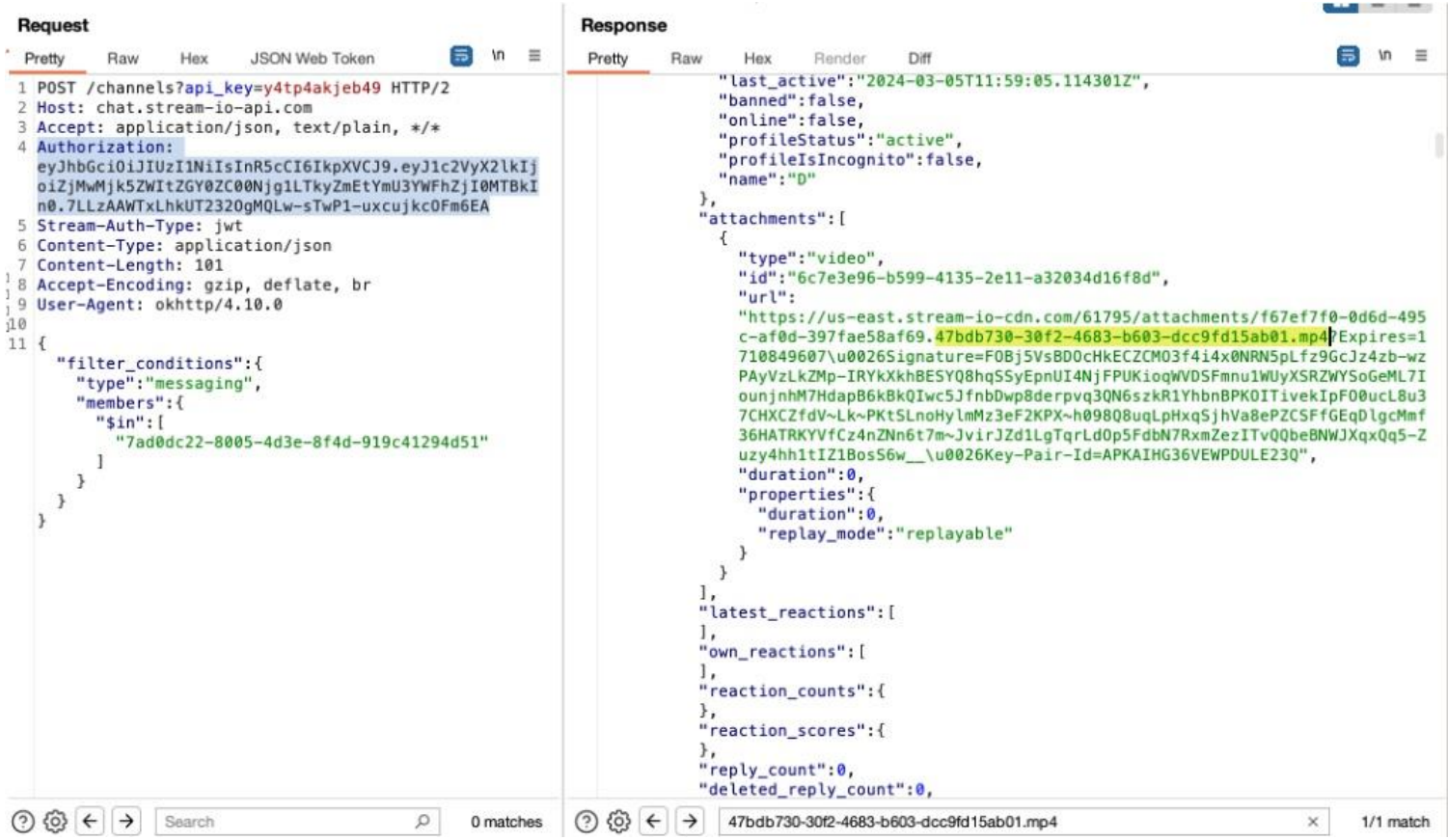
Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 3: Uploading normal video

3.If we again read this chat as the attacker, using the previous vulnerability 'Read other people's messages', we can see the url to the video in the response, as seen below:

We have to replace '\u0026' for '&' in it.

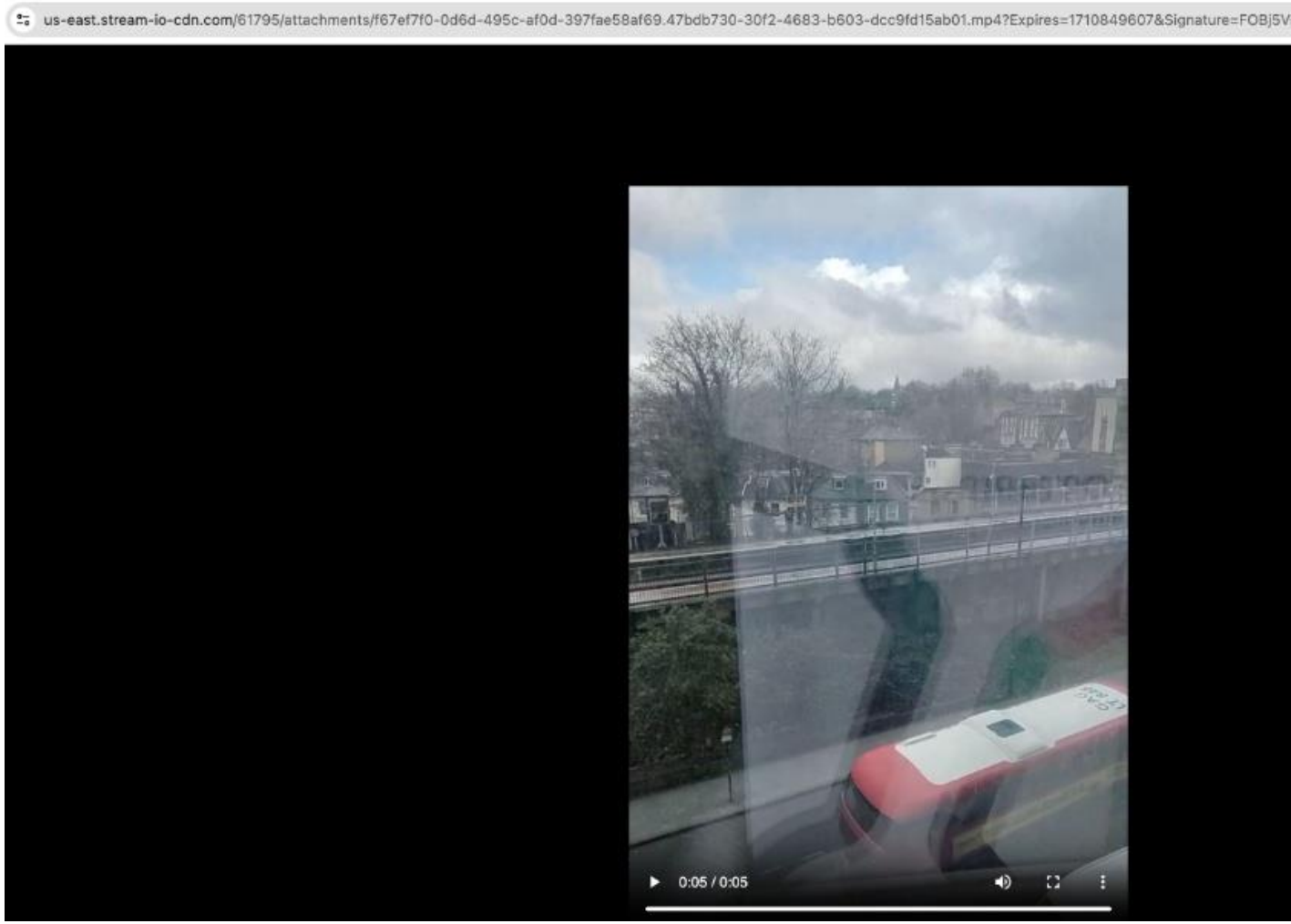


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 3: Uploading normal video

4. Now we can go to the above url unauthenticated.



Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

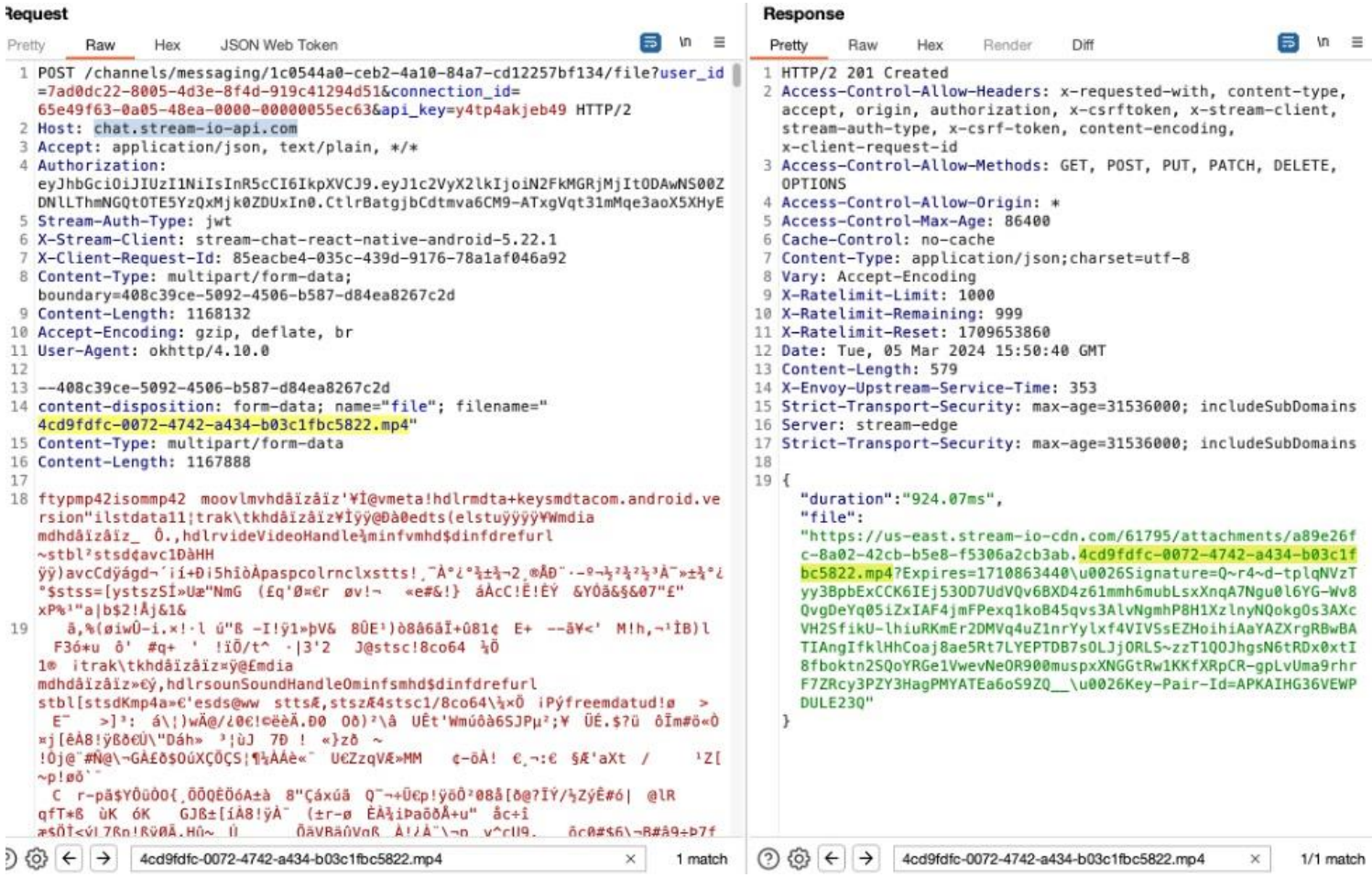
Instance 4: Uploading 'play-once' videos

1.Upload a video, as in instance 3, but set it to 'play once'.

The following requests will be made.

The video will be uploaded to:

chat.stream-io-api.com as seen on the right:

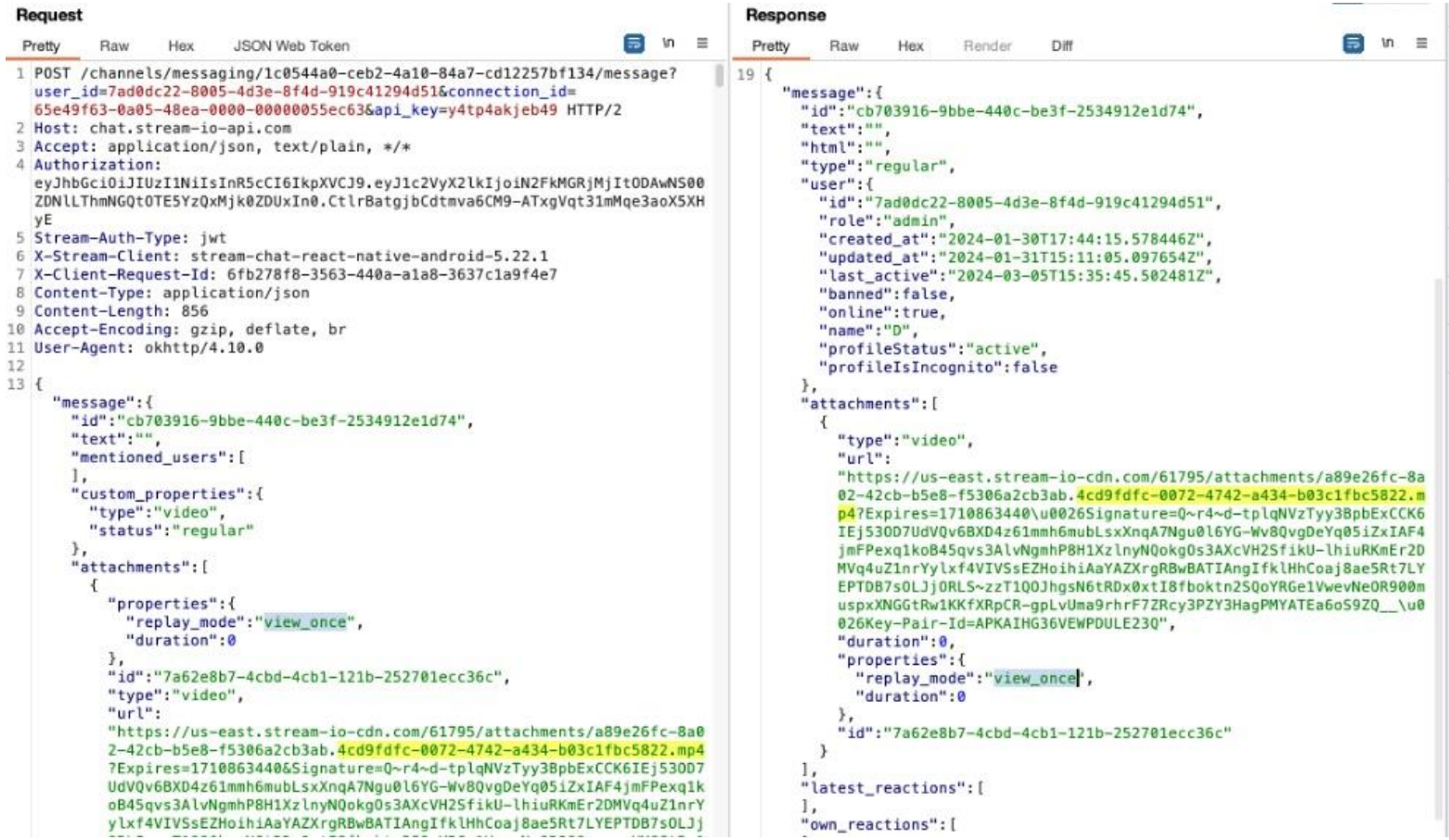


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 4: Uploading 'play-once' videos

2.The returned url will also be sent in the chat in a subsequent request, as seen below:

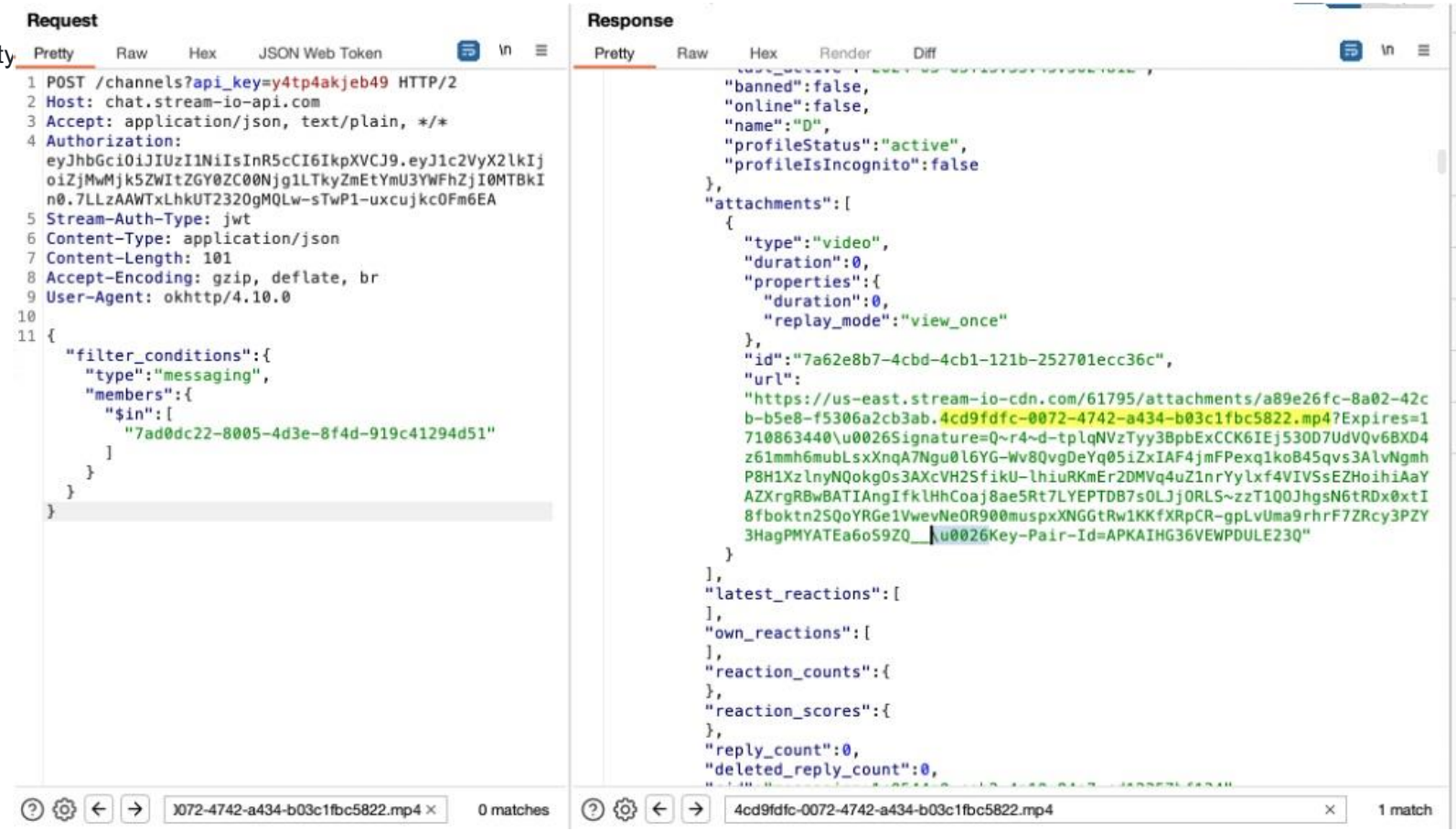


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 4: Uploading 'play-once' videos

3.Now, an attacker can read our chat using the previous vulnerability 'Read other people's chat' and extract this url, as seen below. The url can be extracted from the response and the \u0026 character replaced with &.

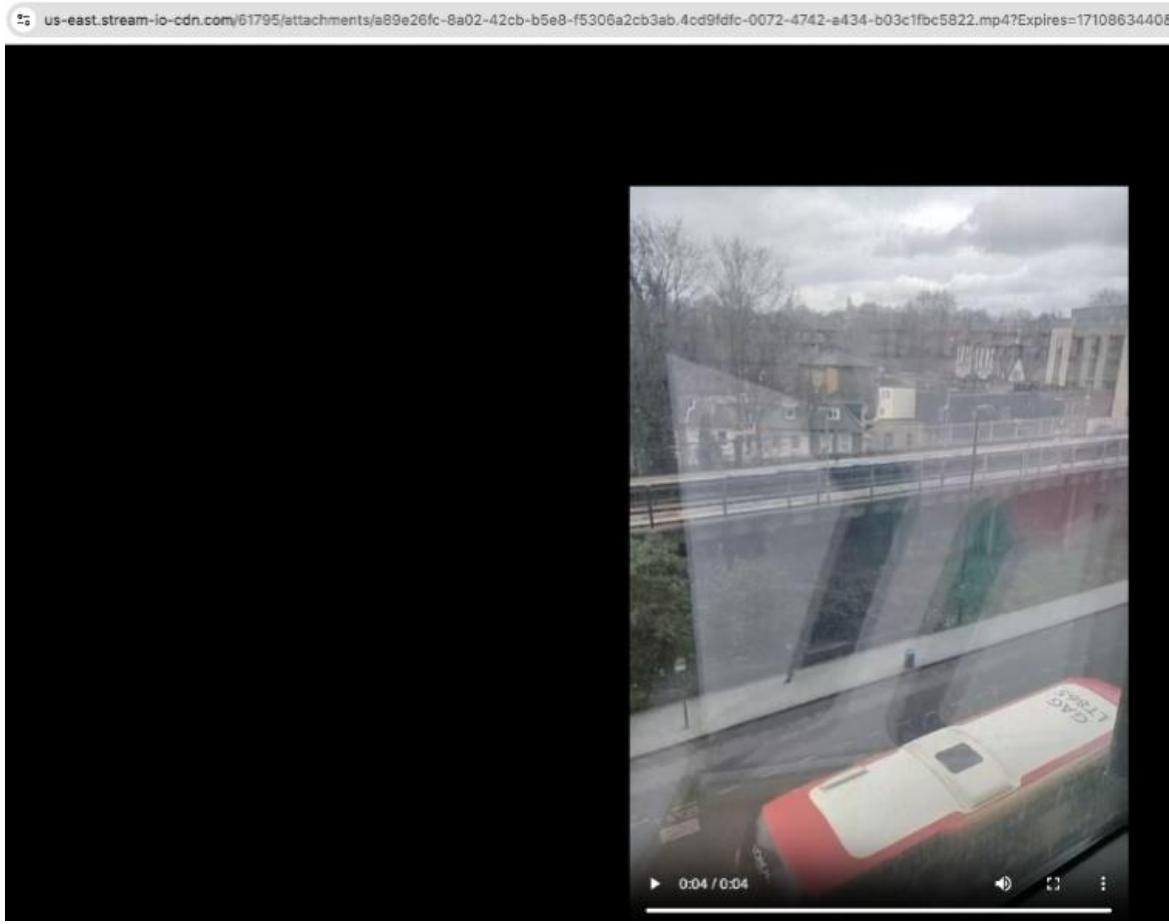


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 4: Uploading 'play-once' videos

4.Thus, we can watch the video unauthenticated and is replay-able:

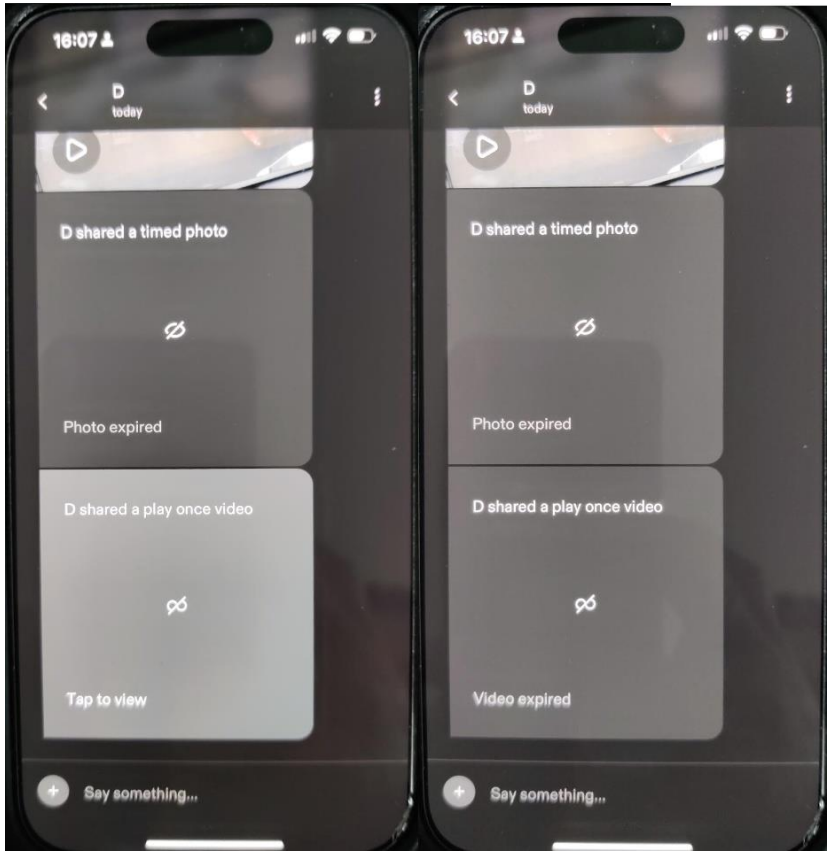


Vulnerability #3 – Unauthenticated access to other people's attachments (photos & videos) from their chats

Reproduction steps:

Instance 4: Uploading 'play-once' videos

5.The receiver of the 'play-once' video, will have no knowledge of the attack. He can still see the video, but only once. After he sees the video, it will say 'video expired'.



'Before' and 'after' the receiver sees the video once.

Vulnerability #4 –Delete, recover and edit other people's messages

#1 Broken Object Level Authorization Category

Details: We discovered that we can recover other people's messages that were deleted in a chat.

In addition, we can edit and delete other people's messages.

In order to do that, we will need the unique 'messageId' value of the message that we want to recover. This is easy to get because when we read our victim's messages, each message has its messageId next to it.

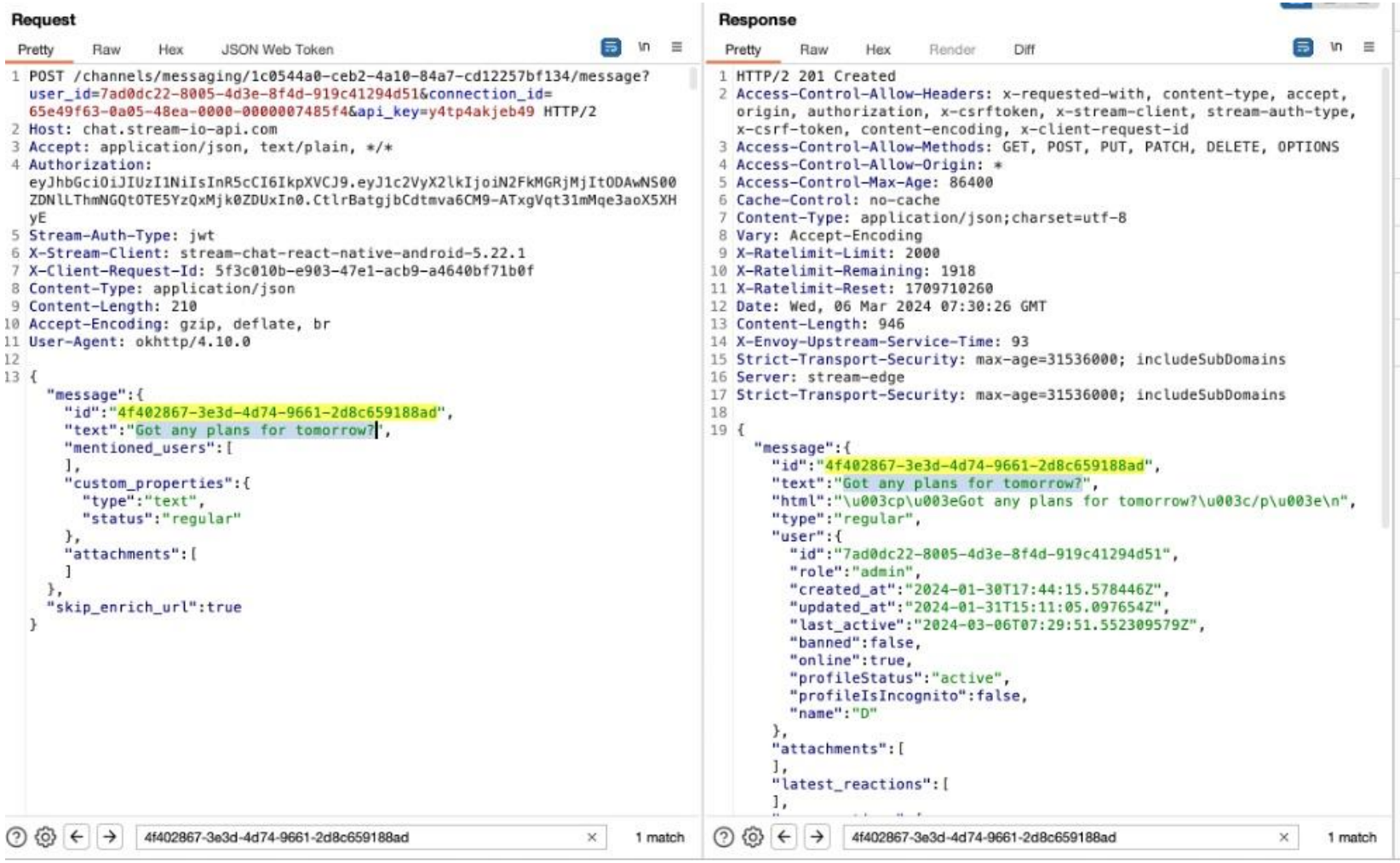
Instance: https://chat.stream-io-api.com/messages/<MessageId>

(Methods: DELETE and PUT)

Vulnerability #4 –Delete, recover and edit other people's messages

Reproduction steps:

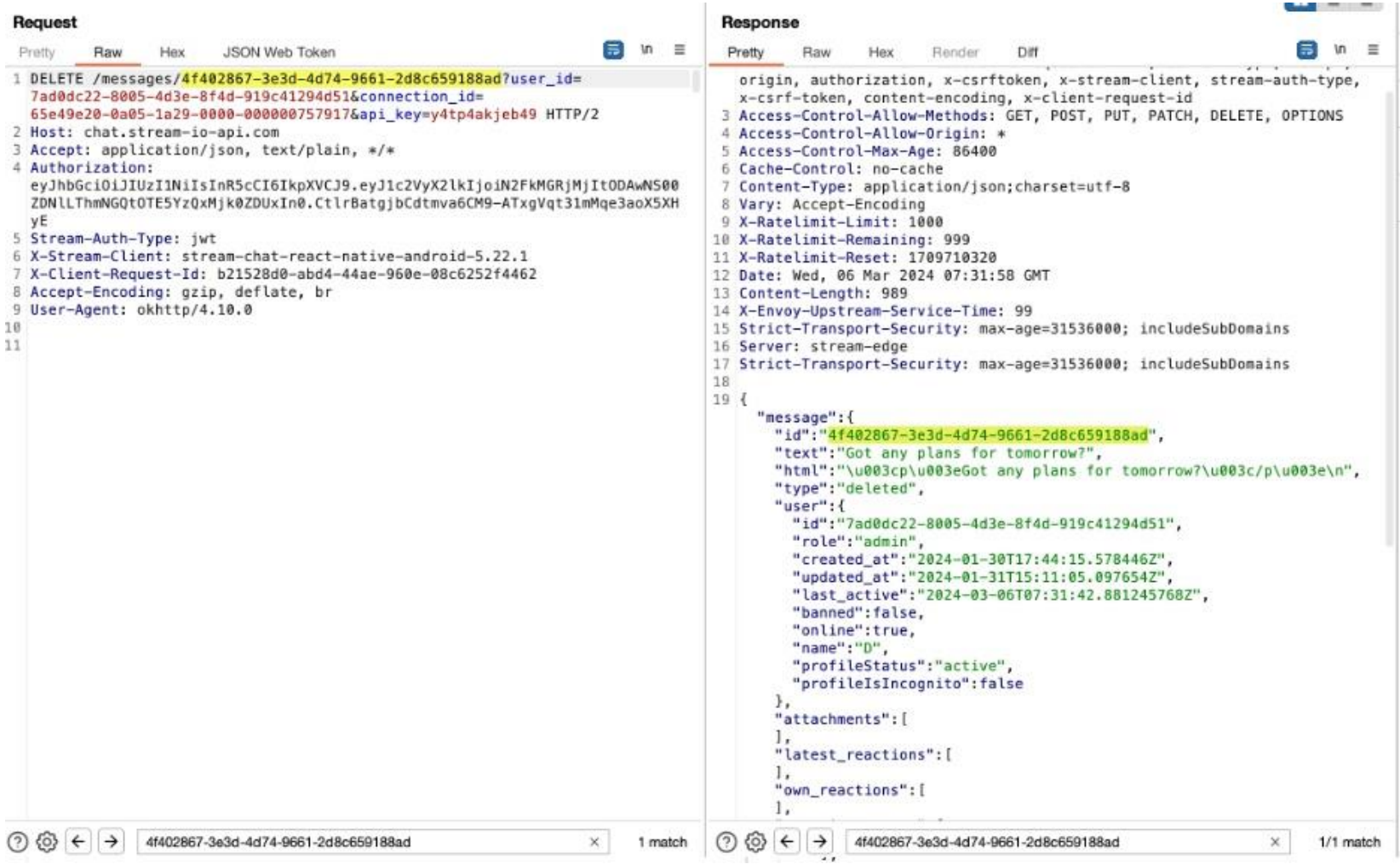
1. Use a proxy tool (Burp) to intercept the traffic.
2. Enter a chat and leave a message to someone:



Vulnerability #4 –Delete, recover and edit other people’s messages

Reproduction steps:

3.Delete the message:

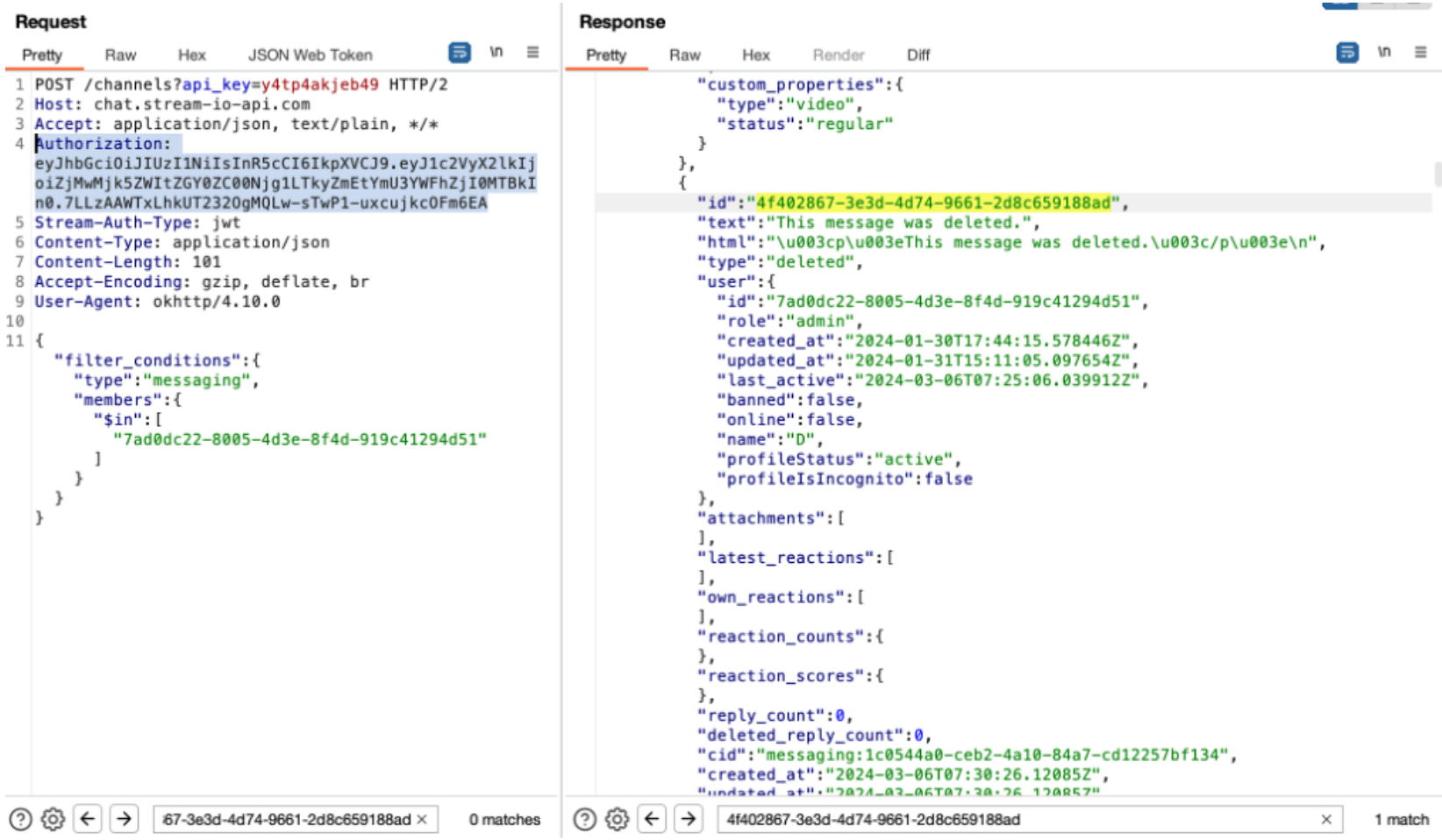


Vulnerability #4 –Delete, recover and edit other people's messages

Reproduction steps:

4.Now let's read the chat as the attacker user using the above vulnerability 'Read other people's messages'.

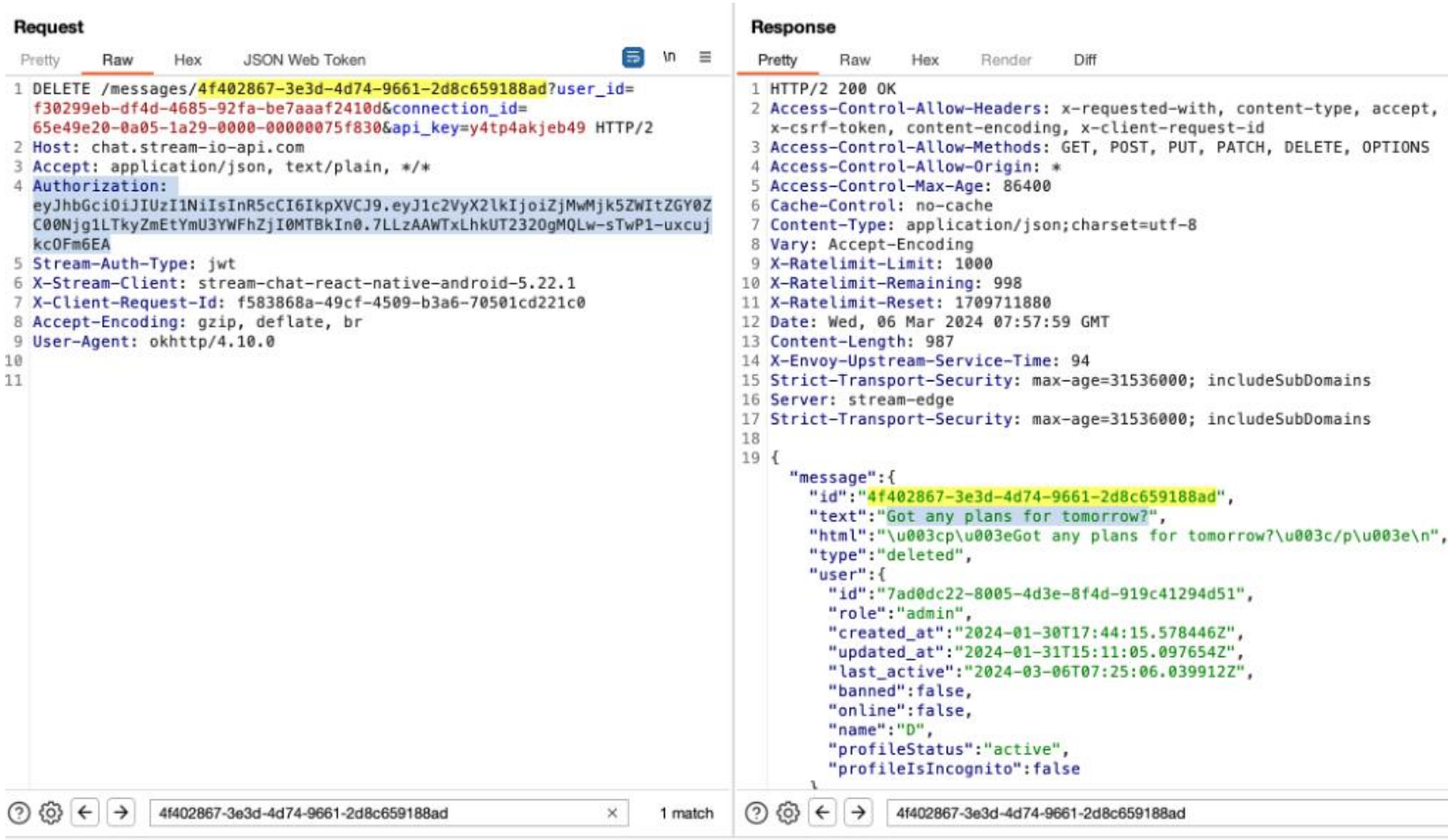
It will say, 'This message was deleted', as seen below:



Vulnerability #4 –Delete, recover and edit other people's messages

Reproduction steps:

5.Now if we call the same DELETE request, as the attacker, we will get back the original message:

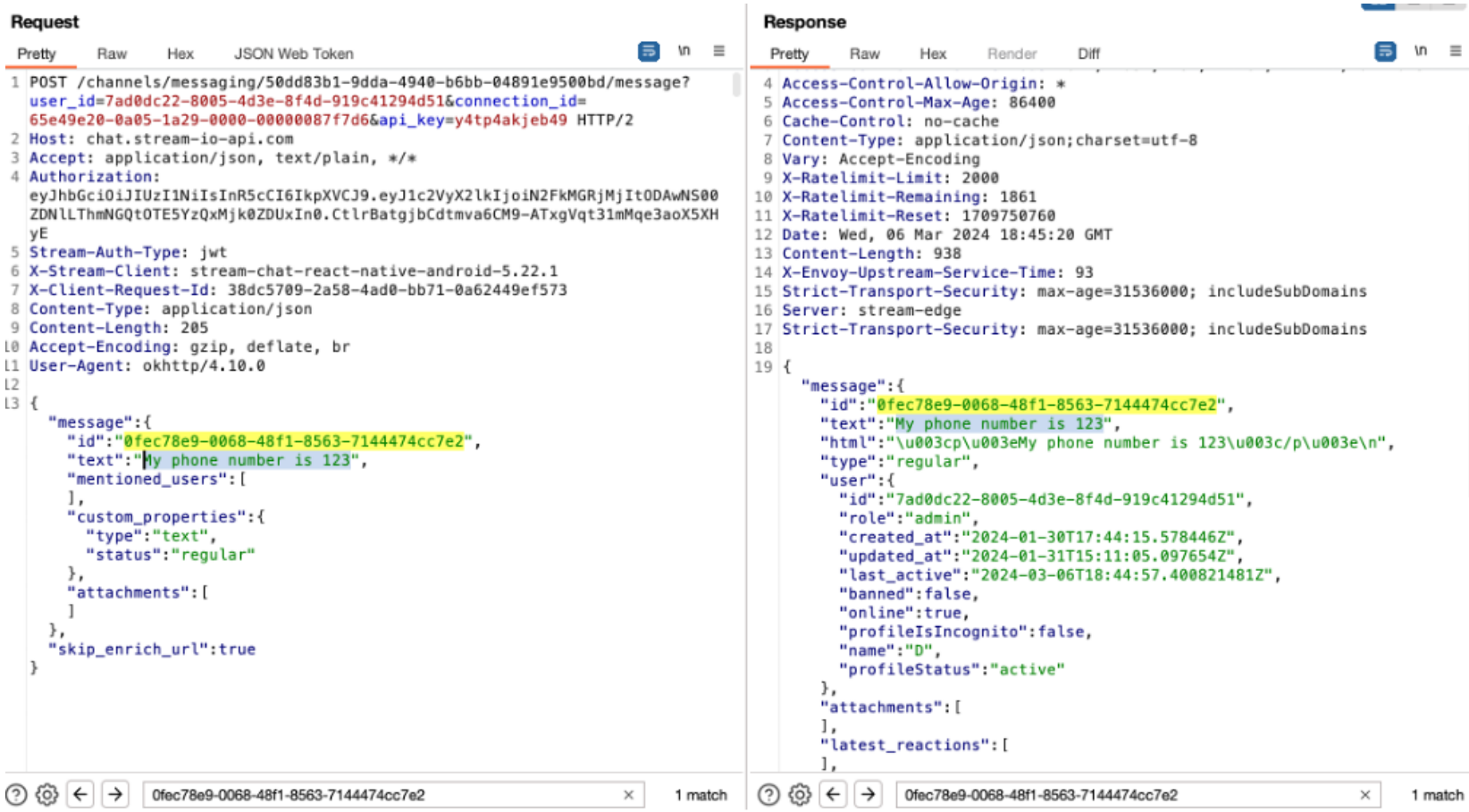


Vulnerability #4 –Delete, recover and edit other people's messages

Reproduction steps:

Instance 2: Edit a message, as a different user than the participants in the chat

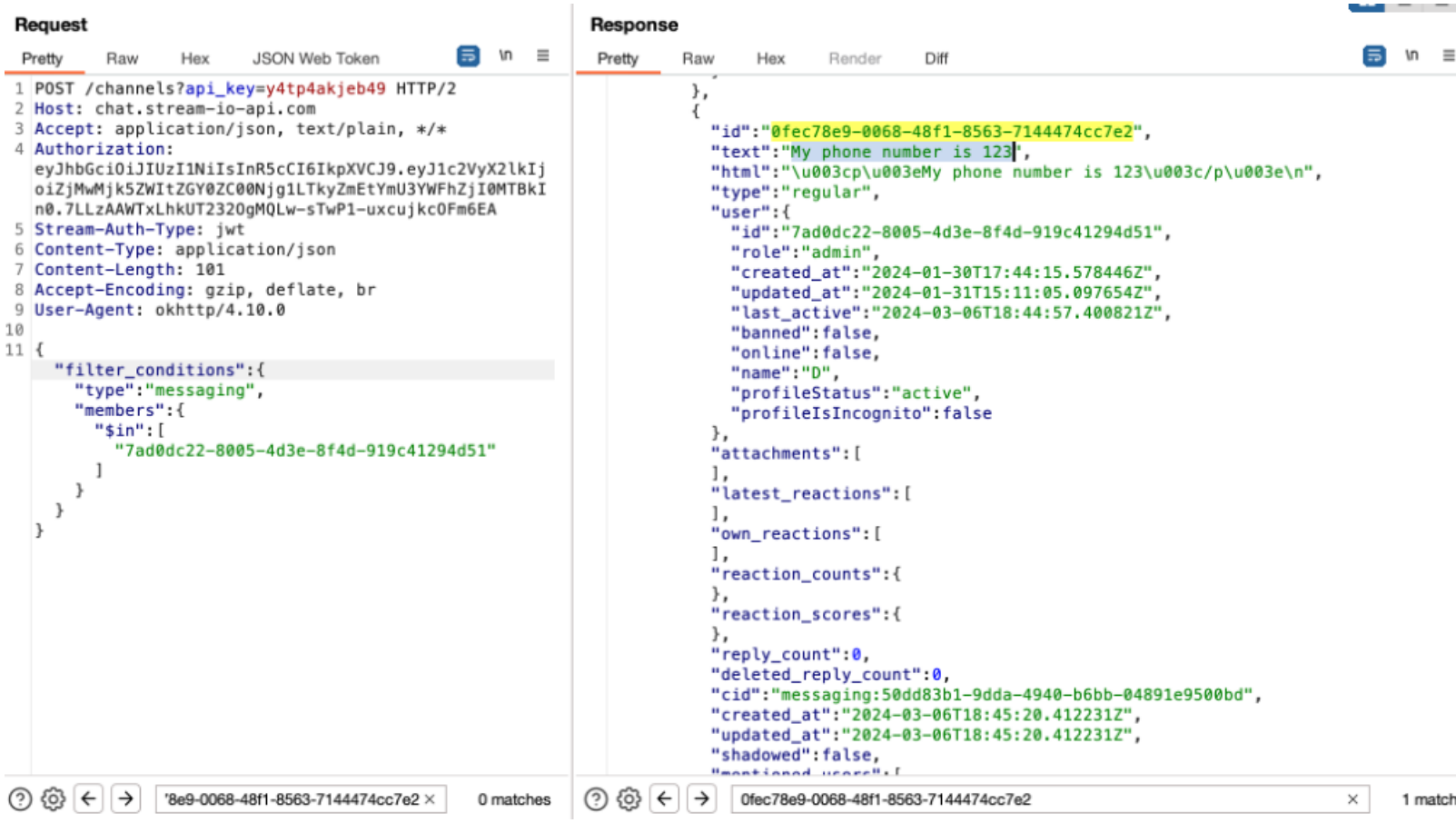
1.First, let's send a message and intercept the request:



Vulnerability #4 –Delete, recover and edit other people’s messages

Reproduction steps:

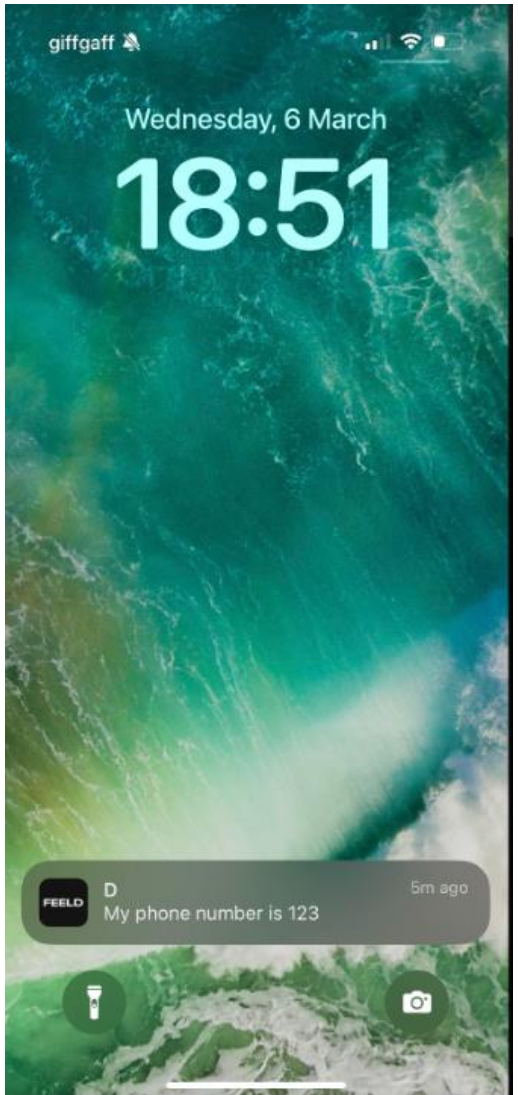
2.And let’s use the previous vulnerability to ‘Read other people’s messages’ as the attacker, in order to find the messageID (‘0fec78e9-0068-48f1-8563-7144474cc7e2’)



Vulnerability #4 –Delete, recover and edit other people’s messages

Reproduction steps:

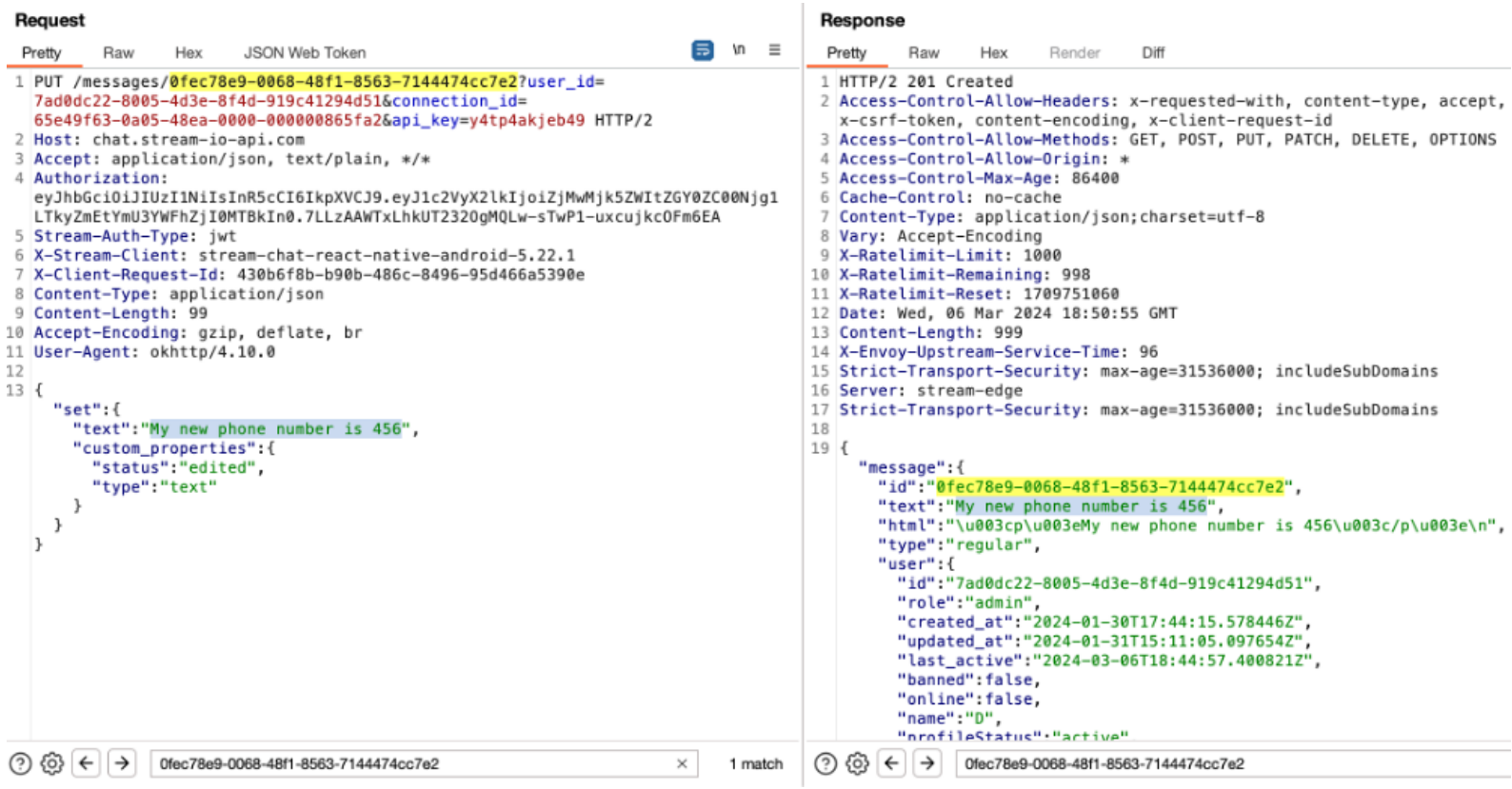
3.The victim will receive a notification:



Vulnerability #4 –Delete, recover and edit other people's messages

Reproduction steps:

4.Edit the message as the attacker, using the messageID and the method PUT on the same endpoint:

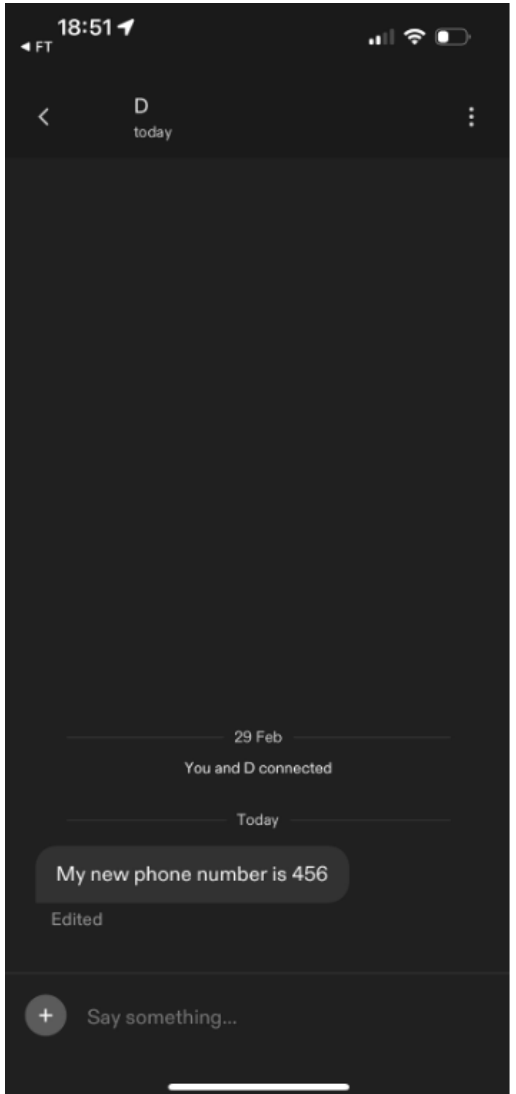


Vulnerability #4 –Delete, recover and edit other people’s messages

Reproduction steps:

5.When the victim taps on the notification from the above step 3, he will see the following message set by the attacker in step 4.

There will be an ‘edited’ sign below the actual message but there are no signs of who did the edit. In addition, every account name is not unique and the attacker could choose any name possible.



Vulnerability #5 – Update someone else's profile information

#1 Broken Object Level Authorization Category

Details: You can update someone else's profile information, including name, sexuality, age, etc.

Instance: https://core.api.feeld.co/graphql ("operationName":"ProfileUpdate")

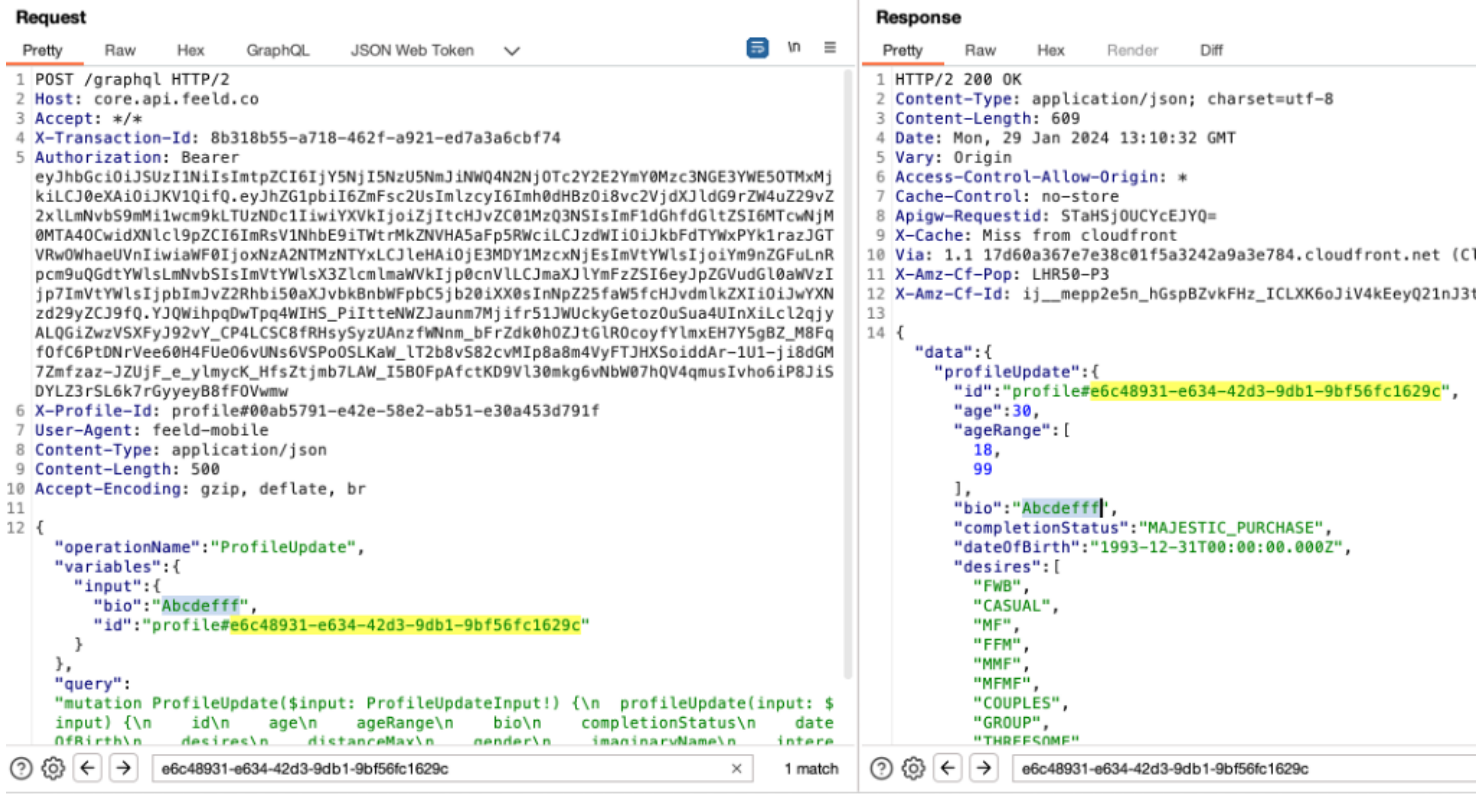
Vulnerability #5 – Update someone else's profile information

Reproduction steps:

1.Let's login the mobile application as the 'attacker' and go to the 'Profile' – 'Edit Profile' menu.

2.Edit 1 thing on the profile such as 'bio', save the change, and intercept the /graphql request with operationName: 'ProfileUpdate'.

3.Modify in the intercepted request the 'id' parameter and add the id of your victim. In addition, add the parameters that you want to update, such as 'bio'.



Vulnerability #6 - Get a ‘Like’ from any user profile

#1 Broken Object Level Authorization Category

Details: You could send ‘Likes’ from profile#2 to profile#3 while logged in as profile#1.

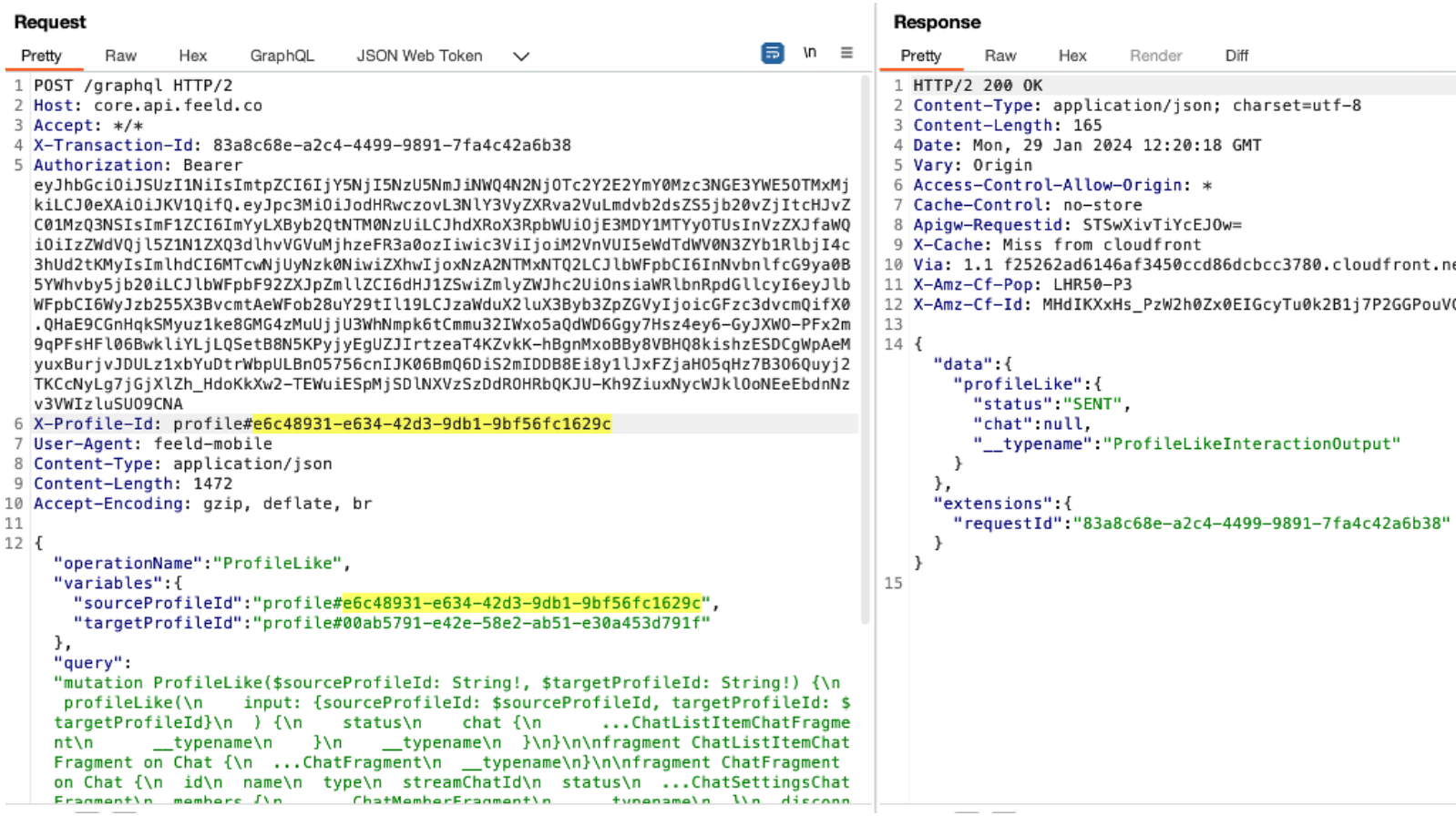
Instance: https://core.api.feeld.co/graphql

(OperationName: ProfileLike)

Vulnerability #6 - Get a ‘Like’ from any user profile

Reproduction steps:

1.Below is a request to send a normal ‘Like’, from user with profileId ending in ‘…9c’ to profileId ‘…1f’, and the successful response:



Request

Pretty Raw Hex GraphQL JSON Web Token

```
POST /graphql HTTP/2
Host: core.api.feeld.co
Accept: */*
X-Transaction-Id: 83a8c68e-a2c4-4499-9891-7fa4c42a6b38
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjY5NjI5NzU5NmJiNWQ4N2NjOTc2Y2E2YmY0Mzc3NGE3YWE5OTMxMjkiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vZjItcHJvZC01MzQ3NSIsImF1ZCI6ImYyLXByb2QtNTM0NzUiLCJhdXRoX3RpbWUiOjE3MDY1MTYyOTUsInVzZXJfaWQiOiIzZWdVQjl5Z1N1ZXQ3dlhvVGVuMjhzeFR3a0ozIiwic3ViIjoiM2VnVUI5eWdTdWV0N3ZYb1RlbjI4c3hUd2tKMyIsImlhdCI6MTcwNjUyNzk0NiwiZXhwIjoxNzA2NTMxNTQ2LCJlbWFpbCI6InNvbnlfcG9ya0B5YWhvby5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmlyZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJzb255X3BvcmtAeWFob28uY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.QHaE9CGnHqkSMyuz1ke8GMG4zMuUjjU3WhNmpk6tCmmu32IWxo5aQdWD6Ggy7Hsz4ey6-GyJXWO-PFx2m9qPFsHFl06BwkliYLjLQSetB8N5KPyjyEgUZJIrtzeaT4KZvkK-hBgnMxoBBy8VBHQ8kishzESDCgWpAeMyuxBurjvJDULz1xbYuDtrWbpULBnO5756cnIJK06BmQ6DiS2mIDDB8Ei8y1lJxFZjaHO5qHz7B3O6Quyj2TKCcNyLg7jGjXlZh_HdoKkXw2-TEWuiESpMjSDlNXVzSzDdROHRbQKJU-Kh9ZiuxNycWJklOoNEeEbdnNzv3VWIzluSUO9CNA
X-Profile-Id: profile#e6c48931-e634-42d3-9db1-9bf56fc1629c
User-Agent: feeld-mobile
Content-Type: application/json
Content-Length: 1472
Accept-Encoding: gzip, deflate, br

{
  "operationName":"ProfileLike",
  "variables":{
    "sourceProfileId":"profile#e6c48931-e634-42d3-9db1-9bf56fc1629c",
    "targetProfileId":"profile#00ab5791-e42e-58e2-ab51-e30a453d791f"
  },
  "query":
  "mutation ProfileLike($sourceProfileId: String!, $targetProfileId: String!) {\n
   profileLike(\n    input: {sourceProfileId: $sourceProfileId, targetProfileId: $
  targetProfileId}\n  ) {\n    status\n    chat {\n      ...ChatListItemChatFragme
  nt\n      __typename\n    }\n    __typename\n  }\n}\n\nfragment ChatListItemChat
  Fragment on Chat {\n  ...ChatFragment\n  __typename\n}\n\nfragment ChatFragment
  on Chat {\n  id\n  name\n  type\n  streamChatId\n  status\n  ...ChatSettingsChat
```

Response

Pretty Raw Hex Render Diff

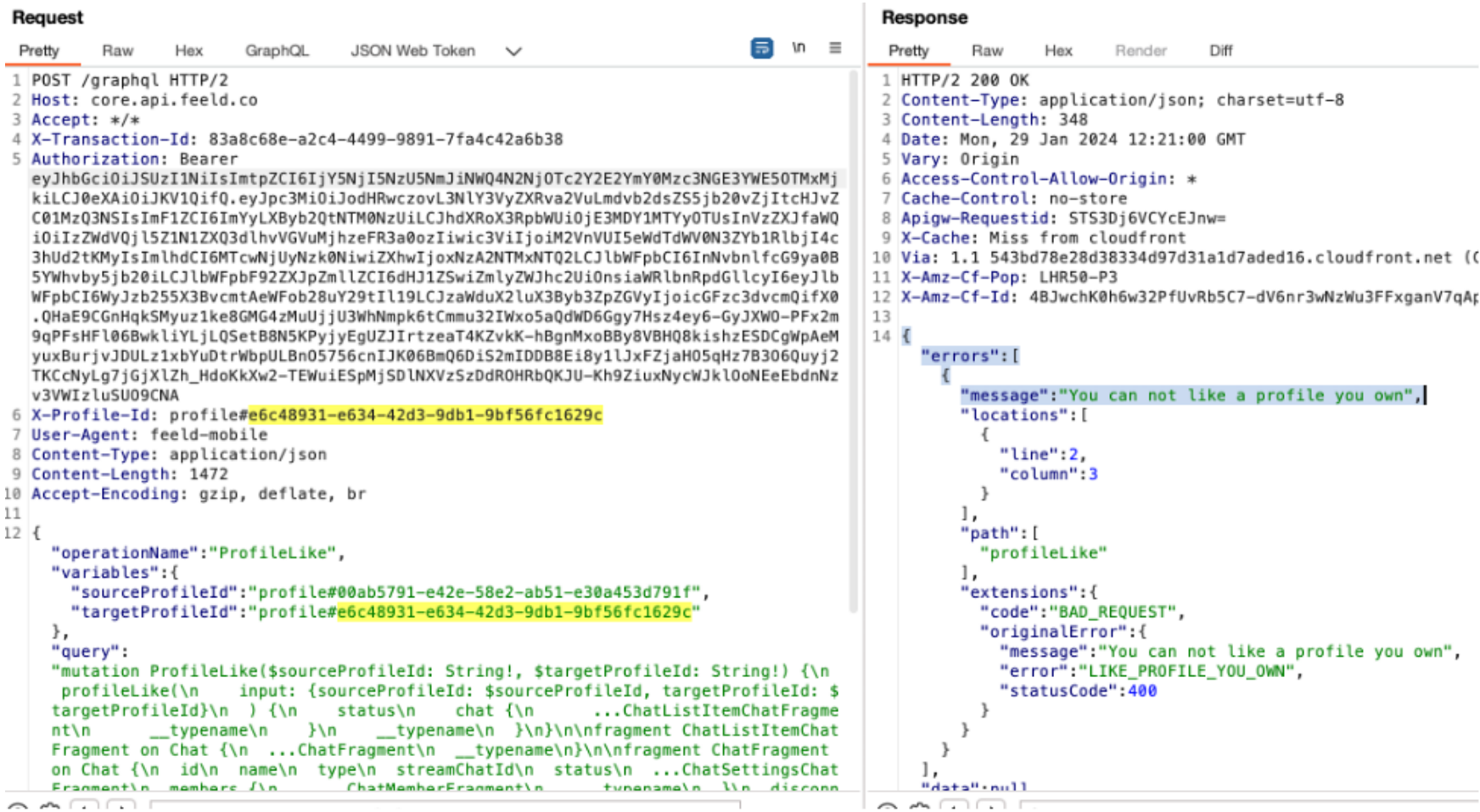
```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 165
Date: Mon, 29 Jan 2024 12:20:18 GMT
Vary: Origin
Access-Control-Allow-Origin: *
Cache-Control: no-store
Apigw-Requestid: STSwXivTiYcEJOw=
X-Cache: Miss from cloudfront
Via: 1.1 f25262ad6146af3450ccd86dcbcc3780.cloudfront.ne
X-Amz-Cf-Pop: LHR50-P3
X-Amz-Cf-Id: MHdIKXxHs_PzW2h0Zx0EIGcyTu0k2B1j7P2GGPouV(

{
  "data":{
    "profileLike":{
      "status":"SENT",
      "chat":null,
      "__typename":"ProfileLikeInteractionOutput"
    }
  },
  "extensions":{
    "requestId":"83a8c68e-a2c4-4499-9891-7fa4c42a6b38"
  }
}
```

Vulnerability #6 - Get a ‘Like’ from any user profile

Reproduction steps:

2.Below is the request and response with a reverse like, from ‘…1f’ to ‘…9c’, which errors:



Request

Pretty Raw Hex GraphQL JSON Web Token

```
POST /graphql HTTP/2
Host: core.api.feeld.co
Accept: */*
X-Transaction-Id: 83a8c68e-a2c4-4499-9891-7fa4c42a6b38
Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6IjY5NjI5NzU5NmJiNWQ4N2NjOTc2Y2E2YmY0Mzc3NGE3YWE5OTMxMj
kiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vZjItcHJvZ
C01MzQ3NSIsImF1ZCI6ImYyLXByb2QtNTM0NzUiLCJhdXRoX3RpbWUiOjE3MDY1MTYyOTUsInVzZXJfaWQ
iOiIzZWdVQjl5Z1N1ZXQ3dlhvVGVuMjhzeFR3a0ozIiwic3ViIjoiM2VnVUI5eWdTdWV0N3ZYb1RlbjI4c
3hUd2tKMyIsImlhdCI6MTcwNjUyNzk0NiwiZXhwIjoxNzA2NTMxNTQ2LCJlbWFpbCI6InNvbnlfcG9ya0B
5YWhvby5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZSwiZmlyZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJlb
WFpbCI6WyJzb255X3BvcmtAeWFob28uY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0
.QHaE9CGnHqkSMyuz1ke8GMG4zMuUjjU3WhNmpk6tCmmu32IWxo5aQdWD6Ggy7Hsz4ey6-GyJXWO-PFx2m
9qPFsHFl06BwkliYLjLQSetB8N5KPyjyEgUZJIrtzeaT4KZvkK-hBgnMxoBBy8VBHQ8kishzESDCgWpAeM
yuxBurjvJDULz1xbYuDtrWbpULBnO5756cnIJK06BmQ6DiS2mIDDB8Ei8y1lJxFZjaHO5qHz7B3O6Quyj2
TKCcNyLg7jGjXlZh_HdoKkXw2-TEWuiESpMjSDlNXVzSzDdROHRbQKJU-Kh9ZiuxNycWJklOoNEeEbdnNz
v3VWIzluSUO9CNA
X-Profile-Id: profile#e6c48931-e634-42d3-9db1-9bf56fc1629c
User-Agent: feeld-mobile
Content-Type: application/json
Content-Length: 1472
Accept-Encoding: gzip, deflate, br

{
  "operationName":"ProfileLike",
  "variables":{
    "sourceProfileId":"profile#00ab5791-e42e-58e2-ab51-e30a453d791f",
    "targetProfileId":"profile#e6c48931-e634-42d3-9db1-9bf56fc1629c"
  },
  "query":
  "mutation ProfileLike($sourceProfileId: String!, $targetProfileId: String!) {\n
   profileLike(\n    input: {sourceProfileId: $sourceProfileId, targetProfileId: $
  targetProfileId}\n  ) {\n    status\n    chat {\n      ...ChatListItemChatFragme
  nt\n      __typename\n    }\n    __typename\n  }\n}\n\nfragment ChatListItemChat
  Fragment on Chat {\n  ...ChatFragment\n  __typename\n}\n\nfragment ChatFragment
  on Chat {\n  id\n  name\n  type\n  streamChatId\n  status\n  ...ChatSettingsChat
```

Response

Pretty Raw Hex Render Diff

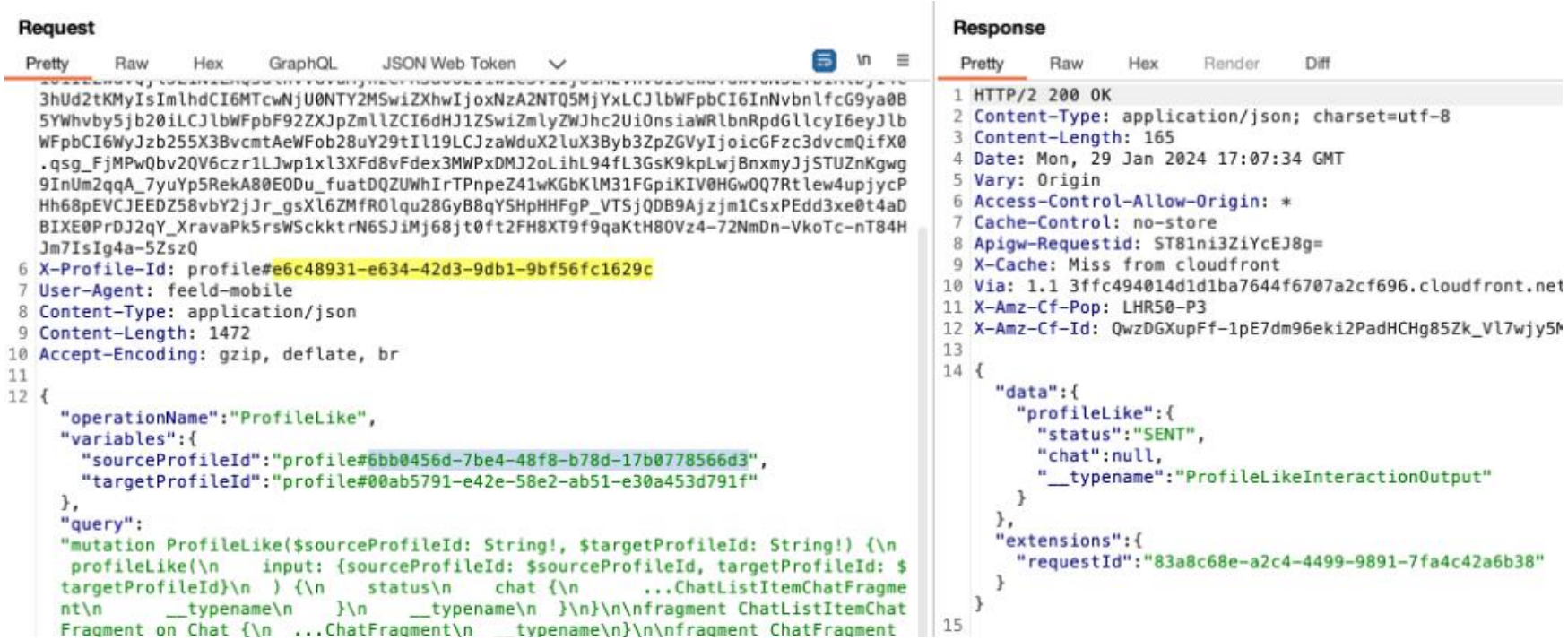
```
HTTP/2 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 348
Date: Mon, 29 Jan 2024 12:21:00 GMT
Vary: Origin
Access-Control-Allow-Origin: *
Cache-Control: no-store
Apigw-Requestid: STS3Dj6VCYcEJnw=
X-Cache: Miss from cloudfront
Via: 1.1 543bd78e28d38334d97d31a1d7aded16.cloudfront.net (C
X-Amz-Cf-Pop: LHR50-P3
X-Amz-Cf-Id: 4BJwchK0h6w32PfUvRb5C7-dV6nr3wNzWu3FFxganV7qAp

{
  "errors":[
    {
      "message":"You can not like a profile you own",
      "locations":[
        {
          "line":2,
          "column":3
        }
      ],
      "path":[
        "profileLike"
      ],
      "extensions":{
        "code":"BAD_REQUEST",
        "originalError":{
          "message":"You can not like a profile you own",
          "error":"LIKE_PROFILE_YOU_OWN",
          "statusCode":400
        }
      }
    }
  ],
  "data":null
```

Vulnerability #6 - Get a ‘Like’ from any user profile

Reproduction steps:

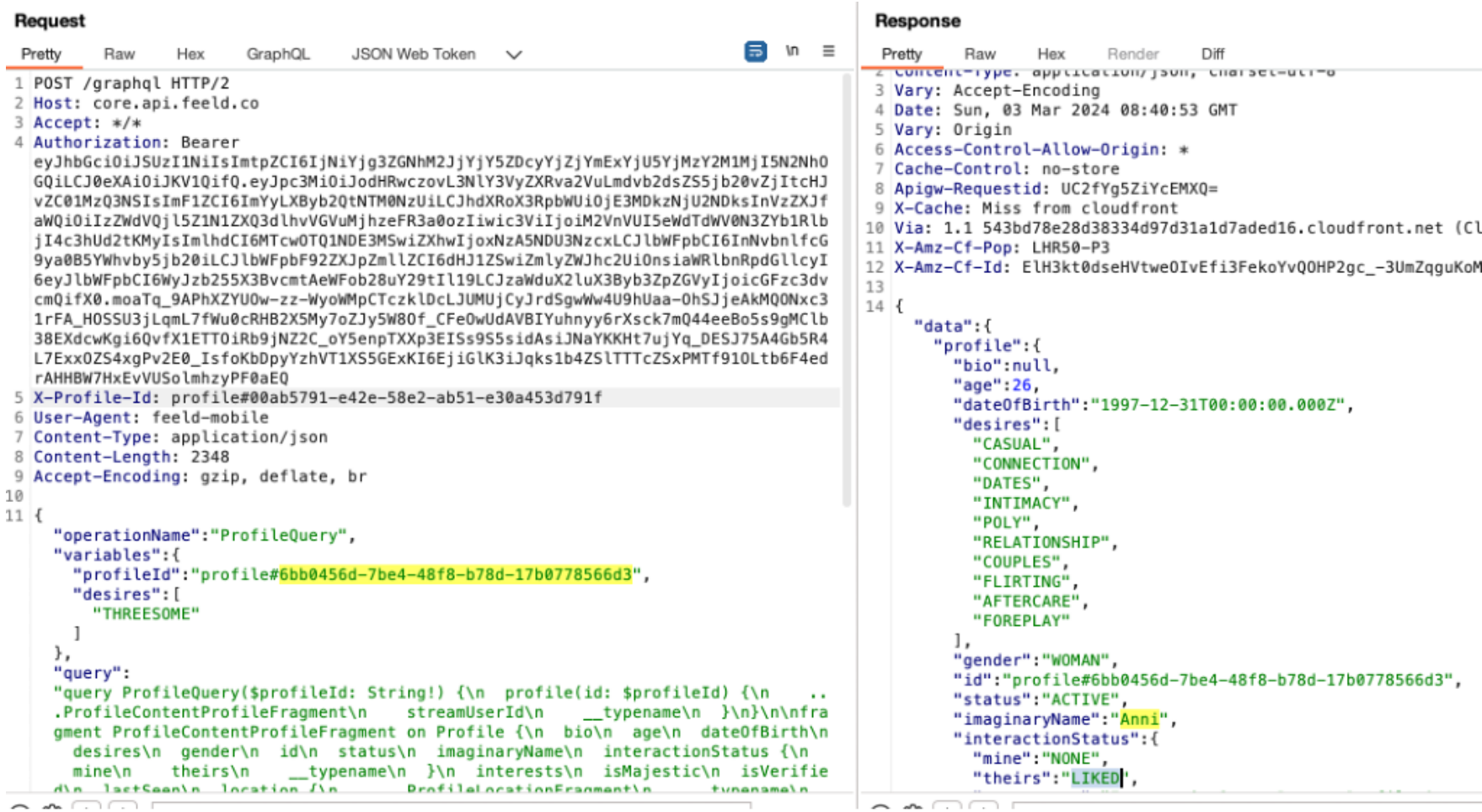
3.Now, send a like from a random profile ‘….d3’ to one of our profiles ‘…1f’ , while logged in as user ‘…9c’:



Vulnerability #6 - Get a ‘Like’ from any user profile

Reproduction steps:

4.Now get the profile details (ImaginaryName) of that user with profileId ‘…d3’:

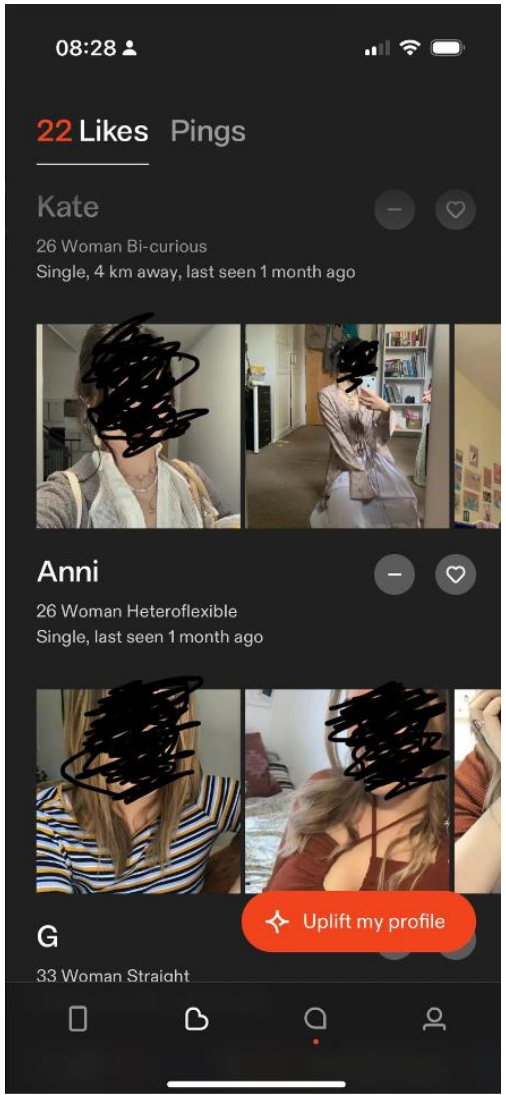


Vulnerability #6 - Get a ‘Like’ from any user profile

Reproduction steps:

5. Now, let’s check our list of likes in the app to see if we received a like from user ‘Anni’.

Given that we have a Premium account, we can view this information in the app. Indeed, we can see that we have received a ‘Like’ from ‘Anni’:



Vulnerability #7 – Send messages in other people's chat

#1 Broken Object Level Authorization Category

Details: We discovered that we can send messages to other people's chats, even though we are not a participant in that chat.

Instance:

https://chat.stream-io-api.com/channels/messaging/<ChannelID>/message

(Method: POST)

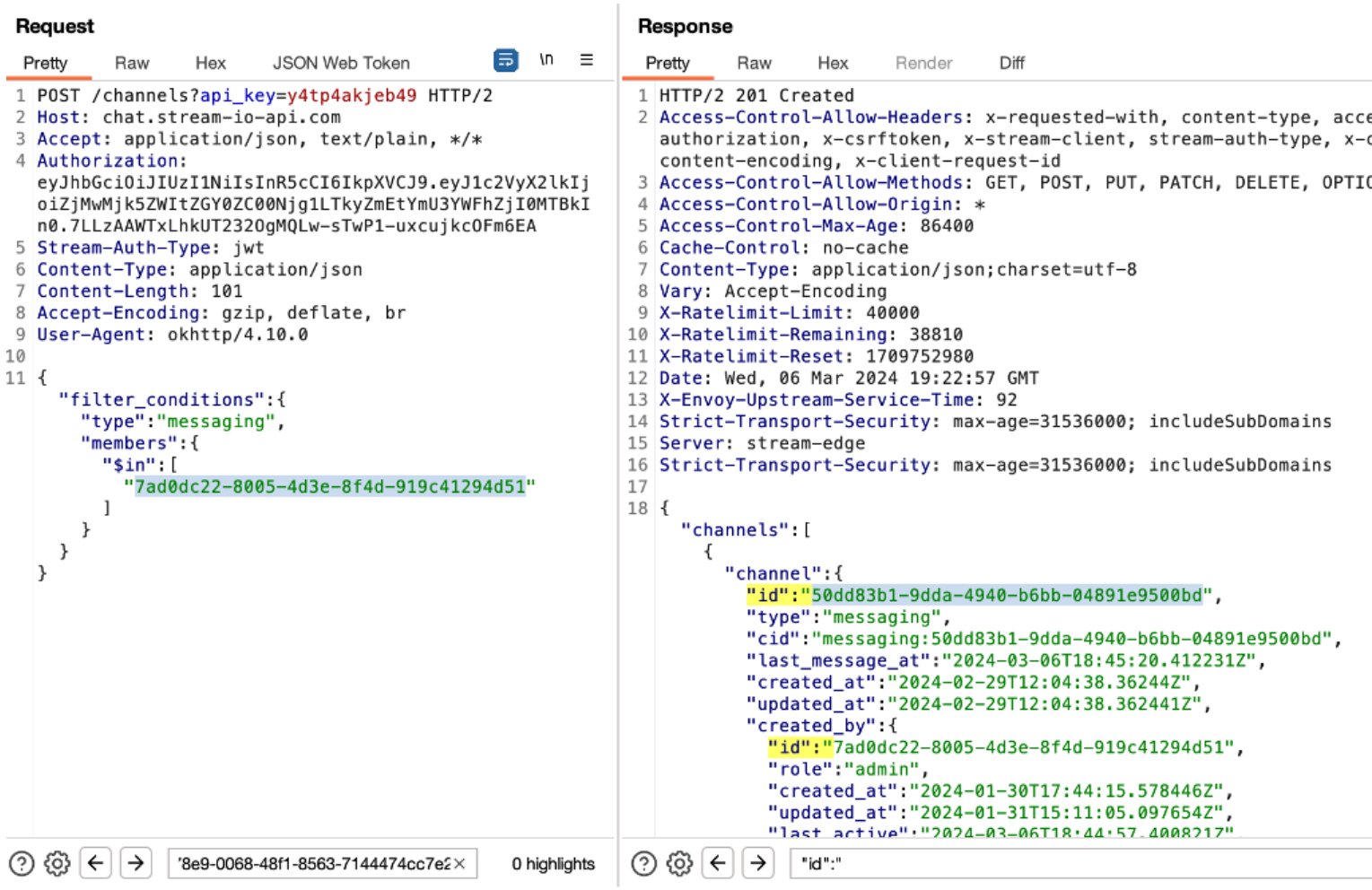
Vulnerability #7 – Send messages in other people's chat

Reproduction steps:

1. Use the previous vulnerability 'Read other people's messages' to find the unique channelID where you want to add your message, such as the the one shown below:

'50dd83b1-9dda-4940-b6bb-04891e9500bd':

Add this channelID to the request path when you exploit this issue in step2 .

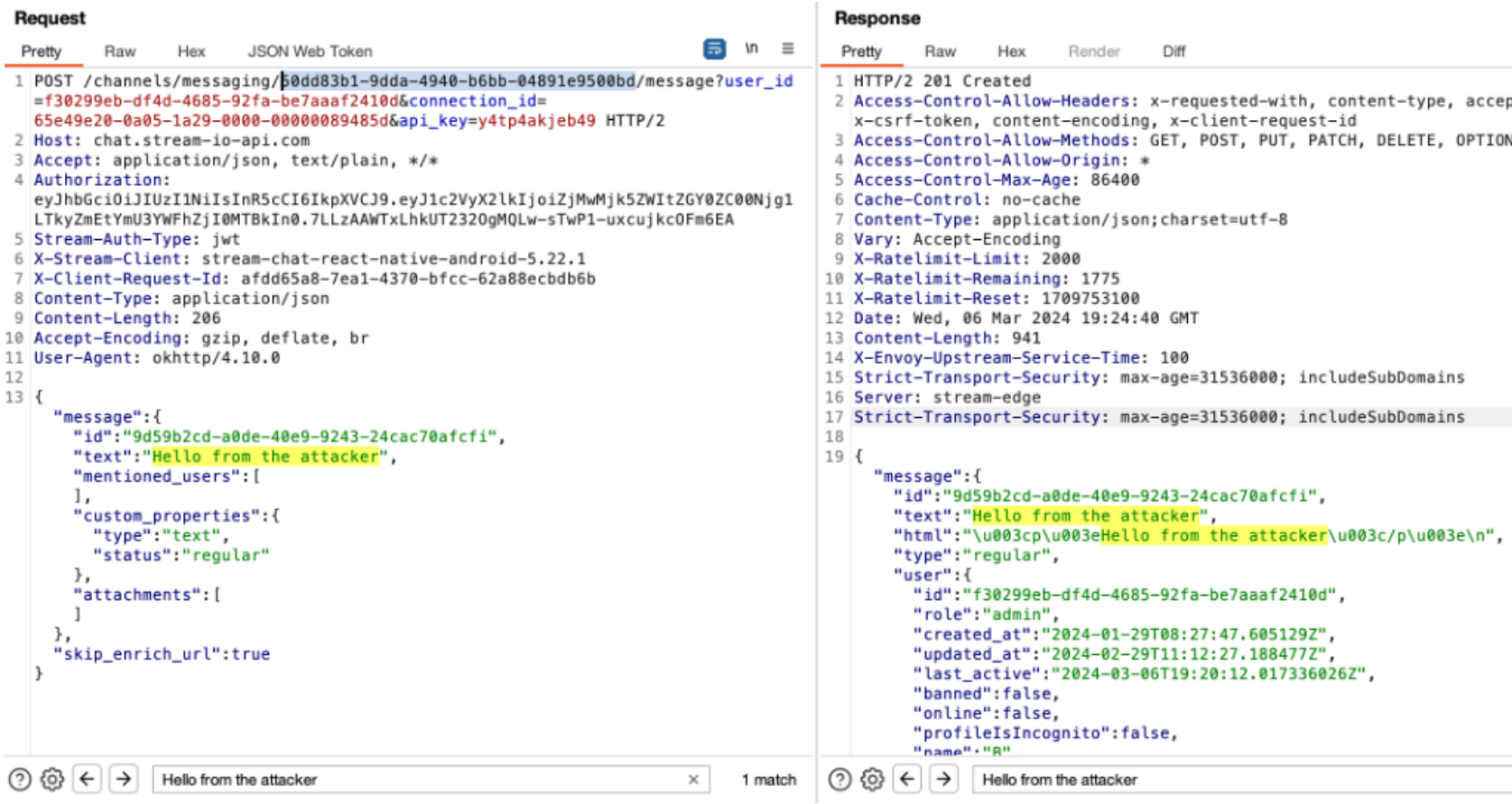


Read other people's messages' and find a 'ChannelID' value

Vulnerability #7 – Send messages in other people's chat

Reproduction steps:

2.Send a message to that channel id:

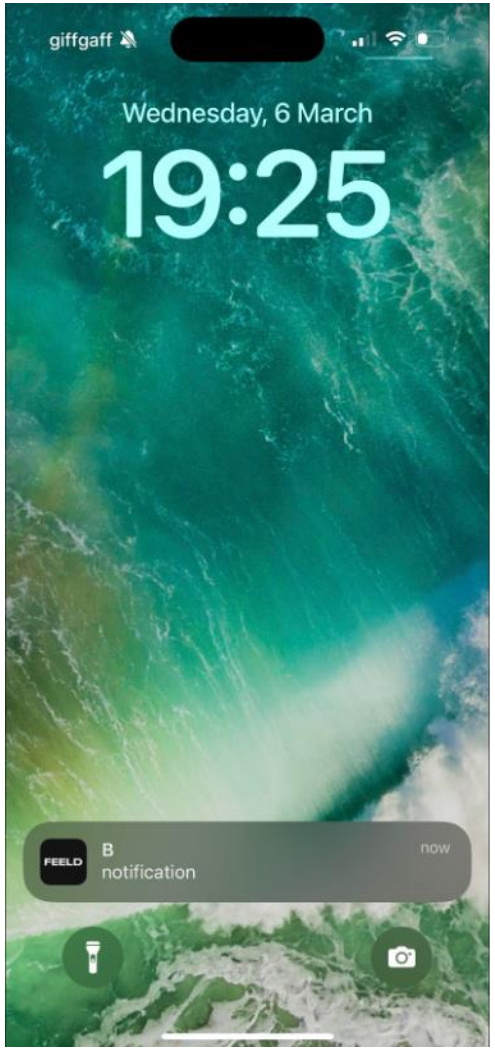


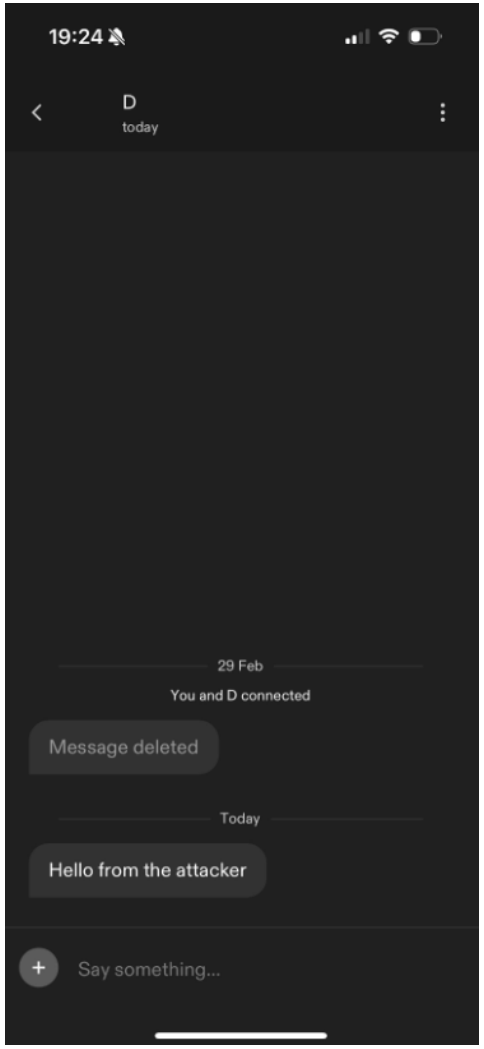
Vulnerability #7 – Send messages in other people's chat

Reproduction steps:

3.The victim will receive a notification, as seen on the right:

Tap the notification and you will see the message. The victim cannot verify whether this message comes from the partner they matched with, or from a 3rd party, an attacker, like in this case.





The chat displayed on the right, is between 2 users: 'D' and 'Bogdan'. Although, the system shows the notification is coming from user 'B' (the attacker's name), the attacker can change their name on their profile, as this field is editable and not unique.

Vulnerability #8 – View other people's matches

#1 Broken Object Level Authorization Category

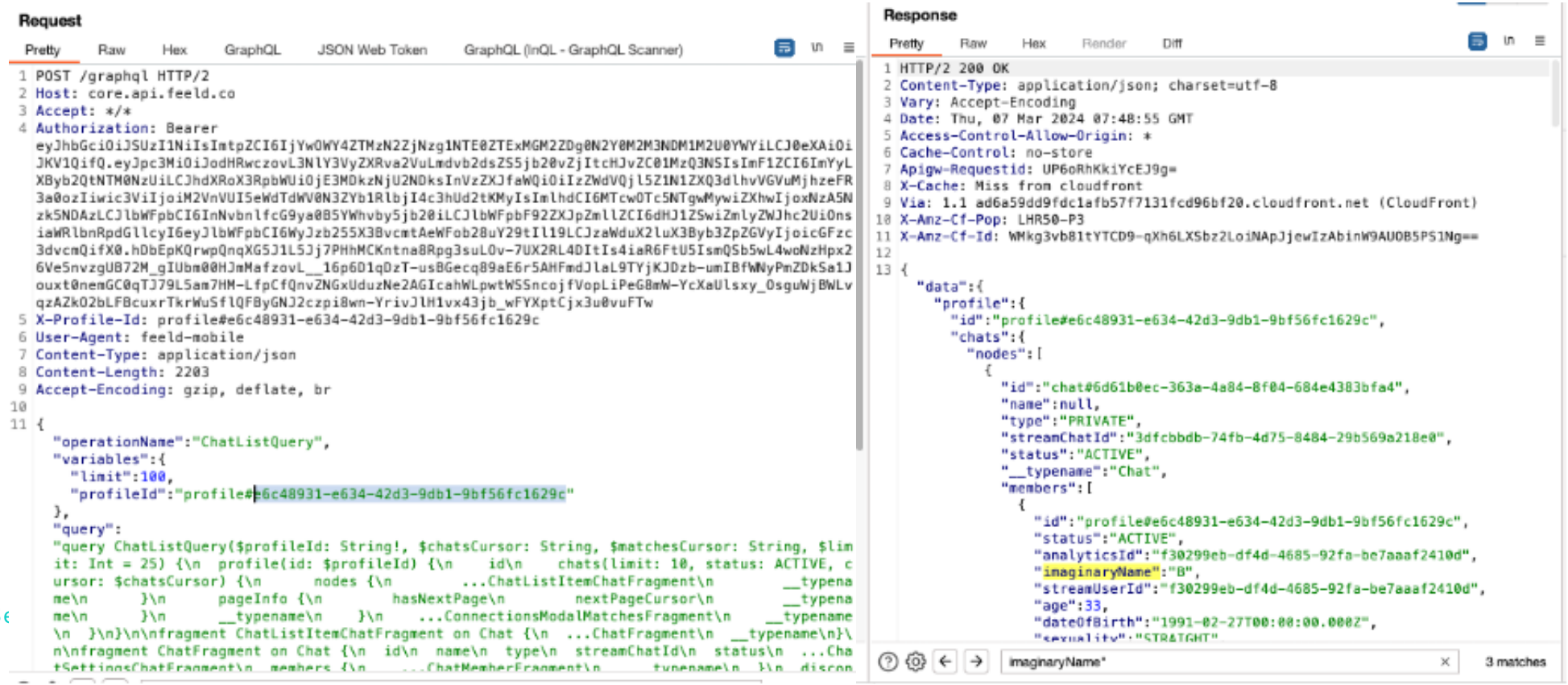
Details: We can check who did other people match with and their full profile information, such as 'imaginaryName', age, photos, gender, sexuality, status, data of birth.

Instance: https://core.api.feeld.co/graphql ("operationName":"ChatListQuery")

Vulnerability #8 – View other people's matches

Reproduction steps:

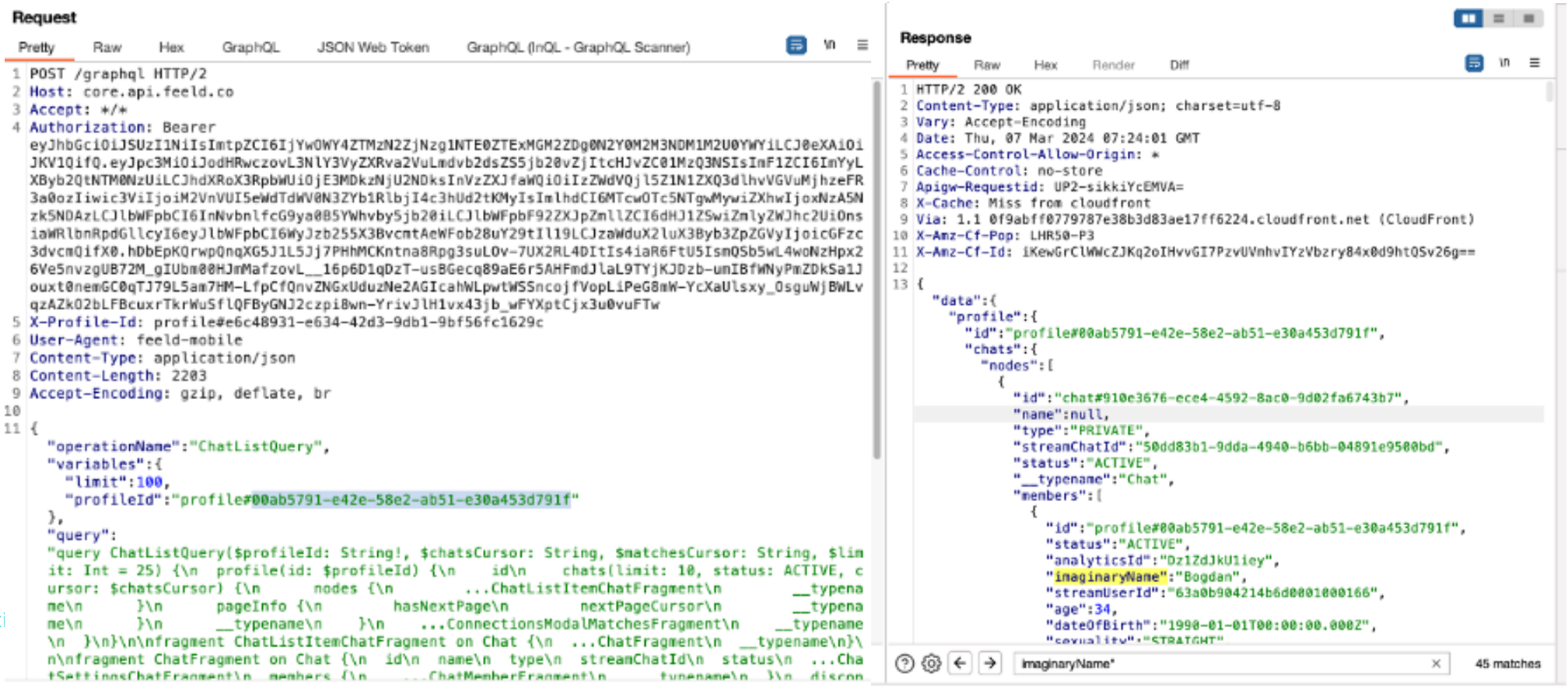
1. Enter the mobile application and go to the 'Discover profiles' menu.
2. It will make a request to /graphql with the "operationName": "ChatListQuery", as seen below:



Vulnerability #8 – View other people's matches

Reproduction steps:

3. Change the profileId to that belonging to a victim user, such as: 00ab5791-e42e-58e2-ab51-e30a453d791f. Thus, we can view that account's matches, as seen below:



Remediation

Developers:

1.Implement the authorization checks. These must be on the back-end and not front-end.

DevSecOps:

1.Integrate security tools in your CICD pipeline.

CISO

1. Do data mapping – identify all personal data your org. collects, processes and stores, including where it is located and how it is used.
2. Implement Data Protection Measures: Introduce technical and organizational measures to protect personal data, such as encryption, access controls, and regular security testing.

Fines:

1. In July 2024, Uber was fined 290million Euros for violating the GDPR's international data transfer rules, by transferring sensitive driver information to its US headquarters

Remediation

Feeld:

Have remediated all the issues we flagged.

Giveaway – Empowering Future AppSec Professionals

3 Burp Suite Certified Exam Vouchers – details on Linkedin @FORTBRIDGE

See Our Leading Research Insights

1. For **web app pentest research** and a peek into PHP internals**,** check **Multiple Concrete CMS Vulnerabilities (Part 1 – RCE)**: This article investigates achieving remote code execution through 2 race conditions vulnerabilities in the file upload functionality in Concrete CMS, p roviding a detailed examination of potential security risks and mitigation strategies.
2. For **API testing research**, check **Mass Account Takeover in Yunmai Smart Scale API**: This article details a pentest of Yunmai's Android and iOS smart scale API, revealing several issues, including a chained attack leading to mass account takeover.
3. For our **open source contribution to security tools,** check **Phishing Like a Pro: A Guide for Pentesters to Add SPF, DMARC, DKIM, and MX Records to Evilginx**: This guide delves into advanced phishing techniques and how to effectively use SPF, DMARC, DKIM, and MX records with Evilginx for penetration testing.

Q & A

THANK YOU!