FINAL Draft

Sequentially Successive Identifiers and Their Relation to Media Exploitation

Bug Report by Kirino

Abstract

I have identified what I believe to be a critical issue in uploaded media's security. When POSTing a status through "v1/statuses" there are no checks performed on potential Media IDs to validate whether the user has access to these images. As Media IDs (I believe) are generated in a sequentially successive and numerical order, it would be extremely simple to re-post every uploaded image to the local instance to a private chat.

This would be done by uploading a test image, catching the id and then looping statuses from 0 up to that ID (or however far back you would wish to go).

Scope Of The Issue

Through testing of the API, I have been able to ascertained six things, which I believe to be factual about the Pleroma BE:

1. Media IDs are sequentially successive in nature and do not employ any randomness to their value.
2. Media IDs are persistent and do not expire
3. It is possible to validate if a user is meant to be able to access a given Media ID
4. Media IDs can not be used to easily determine who uploaded the original record (and in some cases is impossible) by a regular user
5. The POST API request for “v1/statuses” does not utilize this functionality when checking Media IDs
6. Both statuses and chat messages use the same Media ID “counter”.

The first point I inferred by uploading images through the POST API request “v1/media”, in every instance they returned the appropriate number which should follow the previous. I have taken this to mean that Media IDs act in a similar fashion to a counter.

The second point I tested by taking my first post on seal.cafe, which was uploaded April 15, 2023 (https://seal.cafe/notice/AUfI3wH2rkStPoPp0y) and attempting to use the Media ID when submitting a new post (through the POST API request “v1/statuses”), this test was successful and can be seen here.

The third and fourth points were established by testing with the help of another user, when accessing information for a given Media ID (via the GET API request “v1/media/{id}”) you will be met with “Access Denied” if you were not the original uploader of this image. If you did upload the image it will only return a URL and a Description of the image. From this, we can gather that there is functionality in Pleroma to check if you should be able to access a given Media ID and that Media IDs can not be used in the API to easily determine the user who uploaded them.

The fifth and sixth points were established via testing of the following accounts: Flamer and Kirino

A third user on the same instance sent a reaction image to the Flamer account (via a private chat message). Using the GET API request “/api/v1/pleroma/chats/ I was able to fetch the Media ID: “20671324”. I then used the account token for Kirino to publish this private image (of two other users) on my public timeline (via the POST API request “v1/statuses”), this post can be found here.

This shows me that both private chats and public statuses use the same pool of image and that there is no check (on v1/statuses) to ensure a user is allowed to publish the image.

Code Used:

```
from requests import request
from json import loads

class connection:

def __init__(self, url, token):
        self.url = url
        self.token = token



def sendRequest(self, method, url, json=None, files=None):
        req = request(method=method, url=f"{self.url}/{url}", json=json, files=files, cookies=self.token)
        if req.status_code != 200:
                print(req.text)
        req.close()
        return loads(req.text)



def uploadImage(self):
        files = {"file": open("[PATH_TO_IMG]", 'rb')}
        imgID = self.sendRequest(method='POST', url='v1/media', files=files)
        return [imgID["id"]]



def postMessage(self, content, img=None):
        data = {
                "status": content
        }
        if img is not None:
                data.update({"media_ids": img})
        req = self.sendRequest(method='POST', url='v1/statuses', json=data)
        return req


token = {"__Host-pleroma_key": '[PLACE TOKEN HERE]'}
con = connection('[URL]/api', token)

# Using this to get a rough estimate of the latest media id
```

```
print(con.uploadImage())
# Testing an image upload of two other user's private message
con.postMessage("Image test.", ["20671324"])
```

Potential Fixes

As the logic already exists to see if a user was the original uploader of an image, a hot fix would be simply adding that check to “v1/statuses” and any other POST API request which allows image uploads. In the long term, a potential solution would be to check where the Media ID record was first found (either in a status which was set to public or a private status / chat message) and determine whether or not this should be allowed to be uploaded.