 Broken Object Level Authorization (BOLA)

 When authorization controls are lacking or missing, UserA will be able to request UserB’s (along with many other) resources. APIs use values, such as names or numbers, to identify various objects. When we discover these object IDs, we should test to see if we can interact with the resources of other users when unauthenticated or authenticated as a different user. The first step toward exploiting BOLA is to seek out the requests that are the most likely candidates for authorization weaknesses.

 When hunting for BOLA there are three ingredients needed for successful exploitation.

1. Resource ID: a resource identifier will be the value used to specify a unique resource. This could be as simple as a number, but will often be more complicated.
2. Requests that access resources. In order to test if you can access another user's resource, you will need to know the requests that are necessary to obtain resources that your account should not be authorized to access.
3. Missing or flawed access controls. In order to exploit this weakness, the API provider must not have access controls in place. This may seem obvious, but just because resource IDs are predictable, does not mean there is an authorization vulnerability present.

The third item on the list is something that must be tested, while the first two are things that we can seek out in API documentation and within a collection of requests. Once you have the combination of these three ingredients then you should be able to exploit BOLA and gain unauthorized access to resources.

 Finding Resource IDs and Requests

 You can test for authorization weaknesses by understanding how an API’s resources are structured and then attempting to access resources you shouldn’t be able to access. By detecting patterns within API paths and parameters, you might be able to predict other potential resources. The bold resource IDs in the following API requests should catch your attention:

* GET /api/resource/1
* GET /user/account/find?user\_id=15
* POST /company/account/Apple/balance
* POST /admin/pwreset/account/90

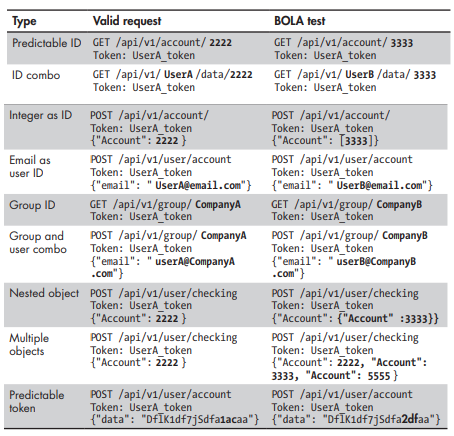
In these instances, you can probably guess other potential resources, like the following, by altering the bold values:

* GET /api/resource/3
* GET /user/account/find?user\_id=23
* POST /company/account/Google/balance
* POST /admin/pwreset/account/111

In these simple examples, you’ve performed an attack by merely replacing the bold items with other numbers or words. If you can successfully access the information you shouldn’t be authorized to access, you have discovered an authorization vulnerability.

Here are a few ideas for

requests that could be good targets for an authorization test.



Let’s check out crAPI and see what sorts of IDs are used to identify resources. We can do this by checking out the documentation and by making requests to the API.

Searching for BOLA

First, let's think about the purpose of our target app and review the documentation. Thinking through the purpose of the app will give you a conceptual overview and help aim your sights. Ask questions like: What can you do with this app? Do you get your own profile? Can you upload files? Do you have an account balance? Is there any part of the app that has data specific to your account? Questions like these will help you search through the available requests and find a starting point for discovering requests that access resources. If we ask these questions about crAPI then you should come up with the following:

* crAPI is an application designed for new vehicle purchases. The app also allows a user to purchase items through a storefront, message other users in a public forum, and update their own user profile.
* Yes, crAPI lets users have their own profile. A user's profile contains a picture, basic user information (name, email, phone number), and a personal video.
* crAPI users have the ability to upload a profile picture and a personal video.
* Parts of the app that are specific to a user's account include
  + The user dashboard with the user's vehicle added
  + The user's profile
  + The user's past orders in the shop
  + The user's posts in the community forum

Now that we have a better idea of the purpose of the app, we should seek out requests that can provide us with relevant resources. If you remember back to the module covering Excessive Data Exposure, then there should be one request that stands out.

* GET /identity/api/v2/videos/:id?video\_id=589320.0688146055
* GET /community/api/v2/community/posts/w4ErxCddX4TcKXbJoBbRMf
* GET /identity/api/v2/vehicle/{resourceID}/location

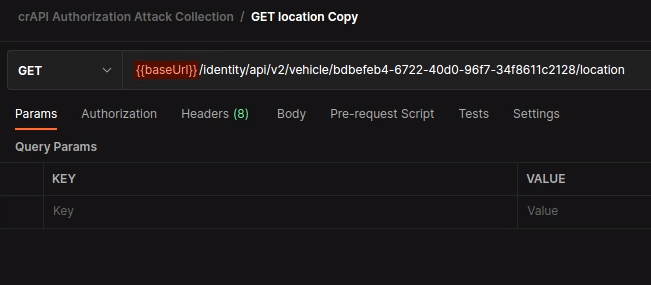
Note that the second request here is for public information. This request retrieves a specific request on the public crAPI forum. As far as BOLA goes, this request has the first two ingredients, but this request functions as designed by sharing public information with a group. So, no authorization is necessary for crAPI users to access this data.

 Authorization Testing Strategy

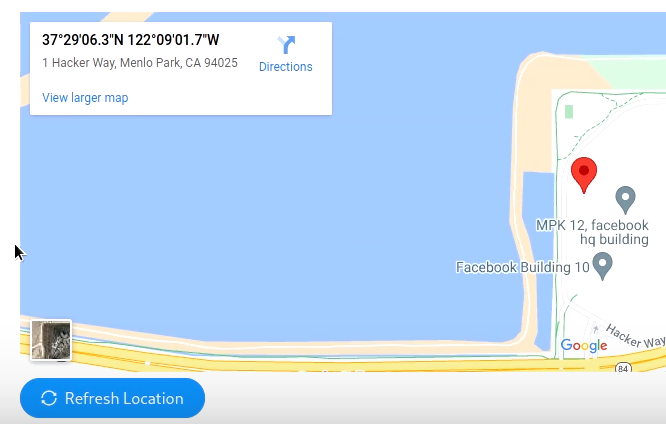
When searching for authorization vulnerabilities the most effective way to find authorization weaknesses is to create two accounts and perform A-B testing. The A-B testing process consists of:

1. Create a UserA account.
2. Use the API and discover requests that involve resource IDs as UserA.
3. Document requests that include resource IDs and should require authorization.
4. Create a UserB account.
5. Obtaining a valid UserB token and attempt to access UserA's resources.

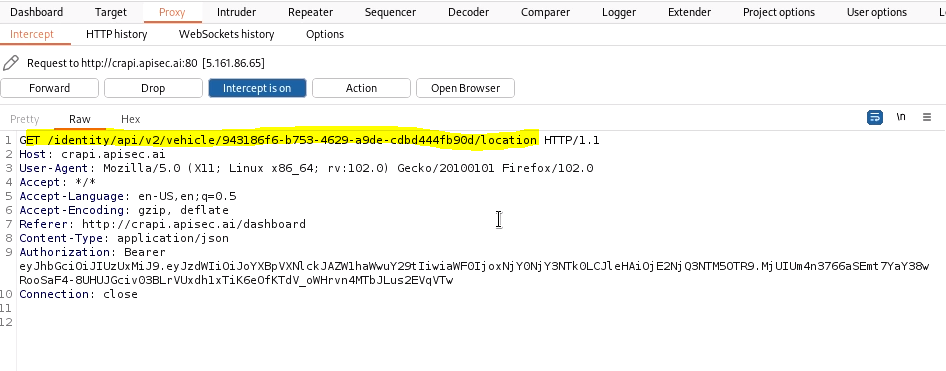
You could also do this by using UserB's resources with a UserA token. In the case of the previously mentioned requests, we should make successful requests as UserA then create a UserB account, update to the UserB token, and attempt to make the requests using UserA's resource IDs. We've already been through the account creation process several times, so I will skip ahead to a request that looks interesting.



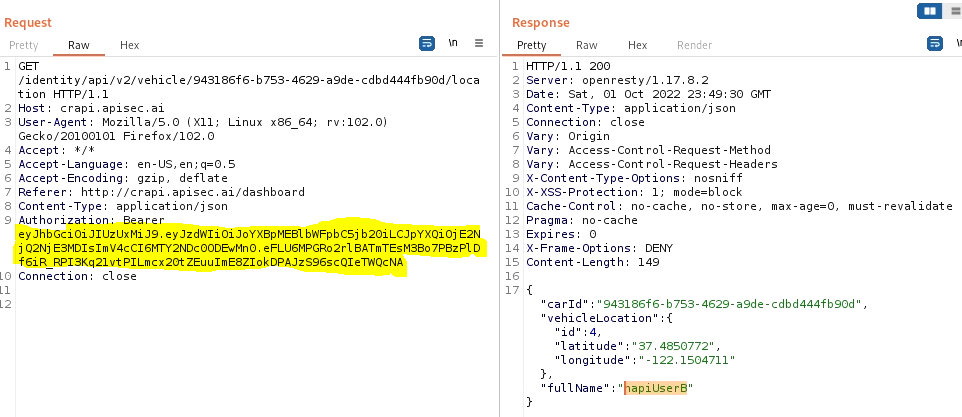
This request looks interesting from a BOLA perspective because it is a request for a location that is based on the complex-looking vehicle ID. As UserB, I've gone through the crAPI interface and registered a vehicle. I then used the "Refresh Location" button on the web app to trigger the above request.



To make things easier for this attack capture the UserB request with Burp Suite.



 Next, perform the BOLA attack by replacing UserB's token with UserA's token and see if you can make a successful request.



Success! UserA's token is able to make a successful request and capture the GPS location of UserB's car along with their vehicleLocation ID and fullName.

In the GET request to the /community/api/v2/community/posts/recent, we discovered that the forum has excessive data exposure. One sensitive piece of data that was exposed was the vehicleID. At first glance, a developer could think that an ID of this complexity (a 32 alphanumeric token) does not require authorization security controls, something along the lines of security through obscurity. However, the complexity of this token would only help prevent or delay a brute-force attack. Leveraging the earlier discovered excessive data exposure vulnerability and combining it with this BOLA vulnerability is a real pro move. It provides a strong PoC and drives home the point of how severe these vulnerabilities really are.