**Improper Assets Management**

[Testing for Improper Assets Management](https://university.apisec.ai/products/api-penetration-testing/categories/2150251354)

In the Analyzing API Endpoints module, we created a Postman collection for crAPI. In this module, we will use this collection to test for Improper Assets Management.

Testing for Improper Assets Management is all about discovering unsupported and non-production versions of an API. Often times an API provider will update services and the newer version of the API will be available over a new path like the following:

* api.target.com/v3
* /api/v2/accounts
* /api/v3/accounts
* /v2/accounts

API versioning could also be maintained as a header:

* *Accept: version=2.0*
* *Accept api-version=3*

In addition versioning could also be set within a query parameter or request body.

* /api/accounts?ver=2
* POST /api/accounts  
    
  {  
  "ver":1.0,  
  "user":"hapihacker"  
  }

In these instances, earlier versions of the API may no longer be patched or updated. Since the older versions lack this support, they may expose the API to additional vulnerabilities. For example, if v3 of an API was updated to fix a vulnerability to injection attacks, then there are good odds that requests that involve v1 and v2 may still be vulnerable.

Non-production versions of an API include any version of the API that was not meant for end-user consumption. Non-production versions could include:

* api.test.target.com
* api.uat.target.com
* beta.api.com
* /api/private
* /api/partner
* /api/test

The discovery of non-production versions of an API might not be treated with the same security controls as the production version. Once we have discovered an unsupported version of the API, we will test for additional weaknesses. Similar to unsupported software vulnerabilities, improper assets management vulnerabilities are an indication that there is a greater chance for weaknesses to be present. Finding versions that are not included in API documentation will be at best a vulnerability for insufficient technical documentation ([CWE-1059](https://cwe.mitre.org/data/definitions/1059.html)) and at worst a gateway to more severe findings and the compromise of the provider.

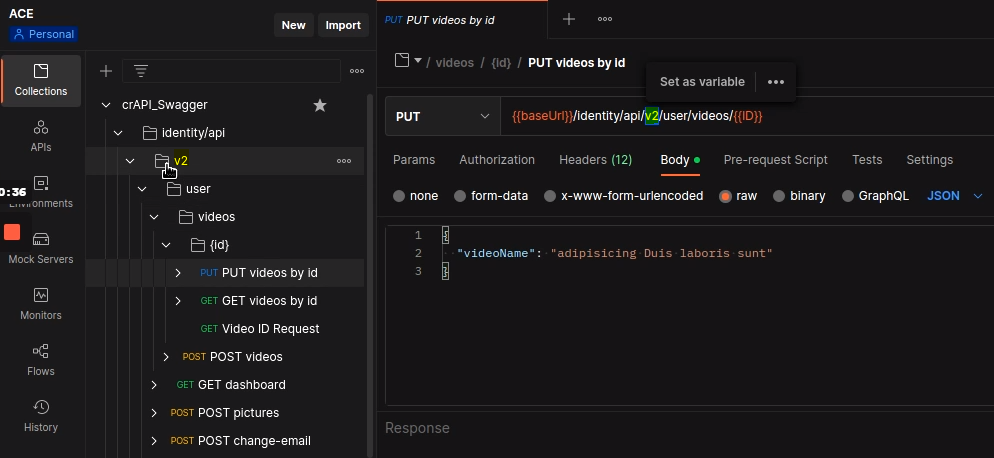
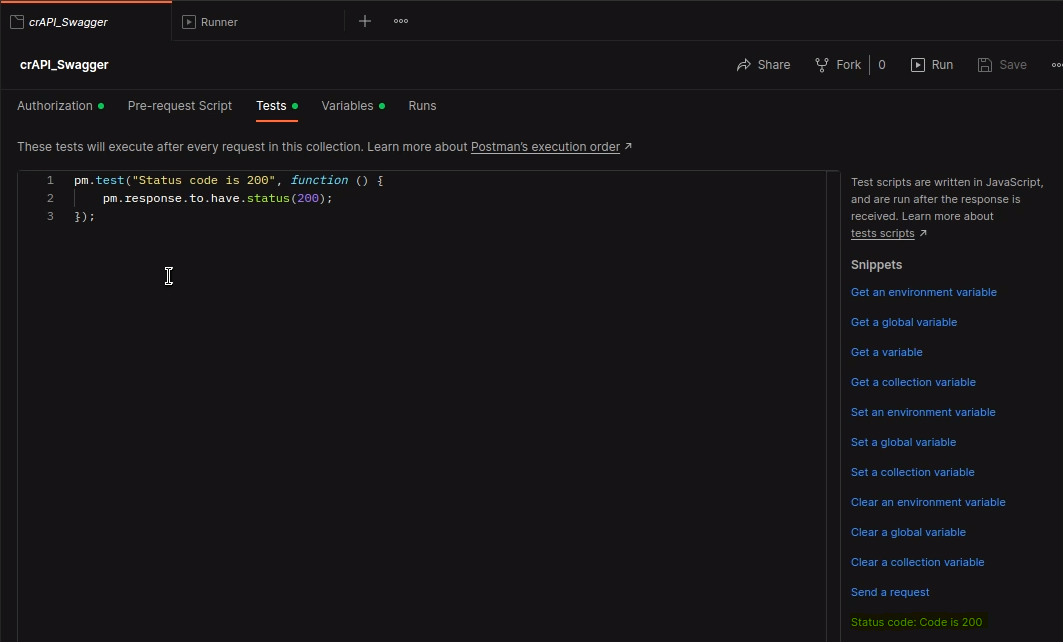
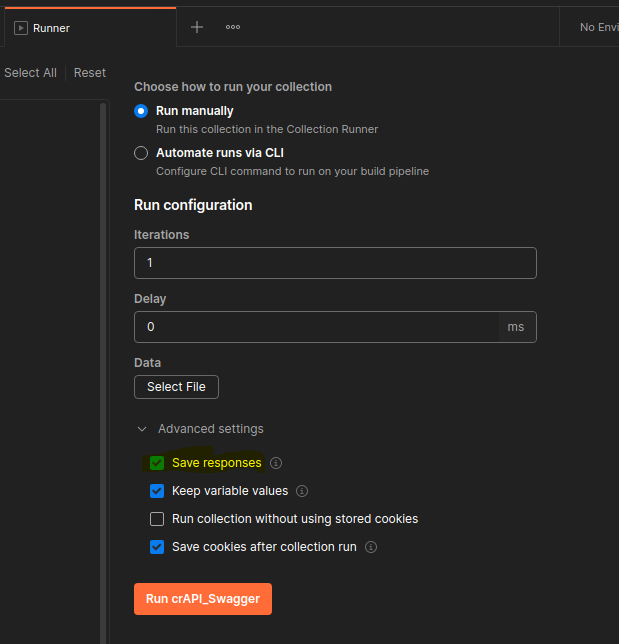
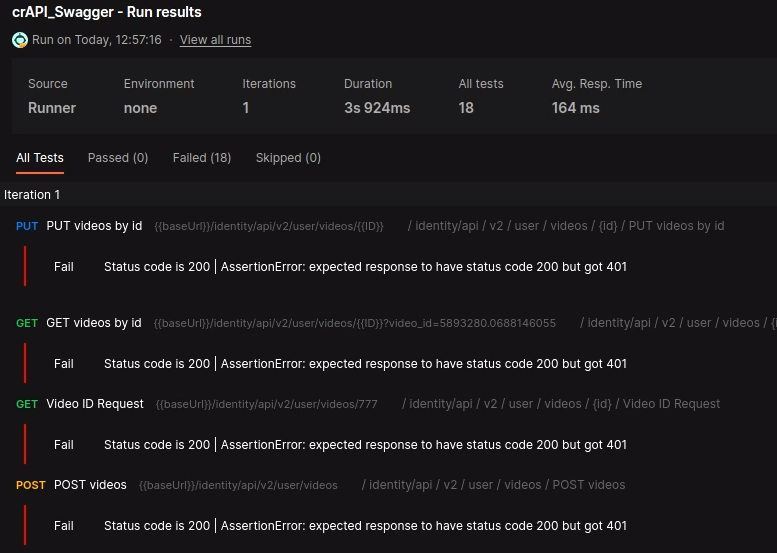
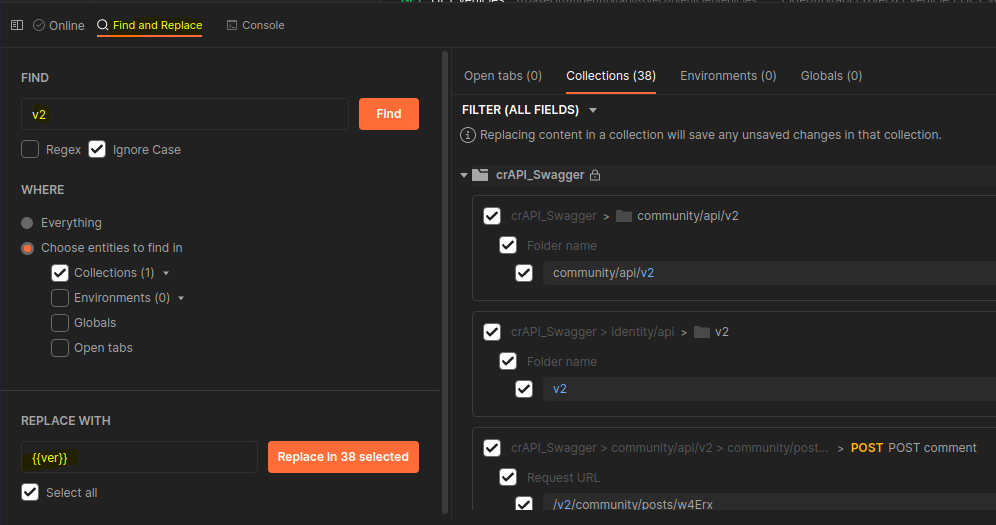
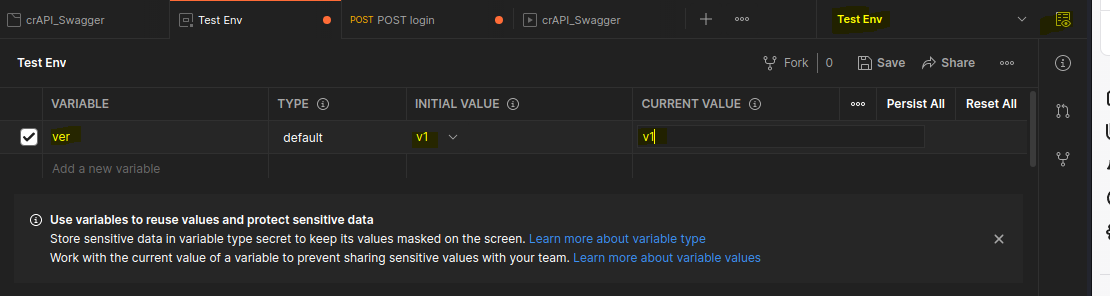
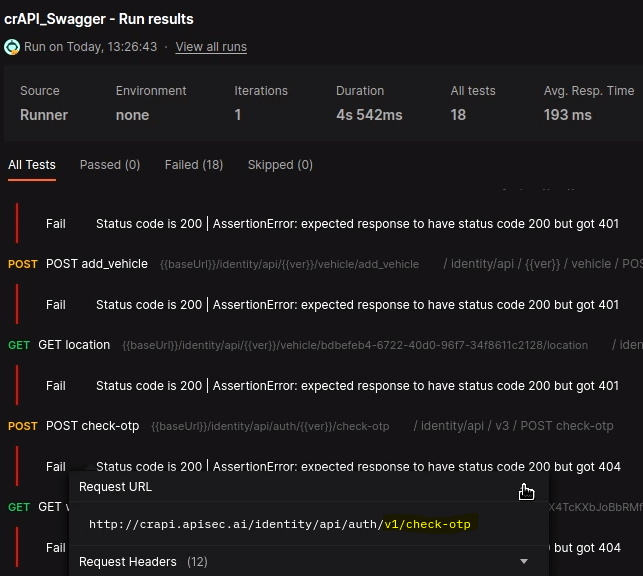
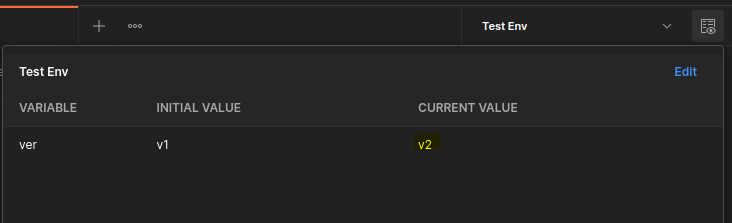
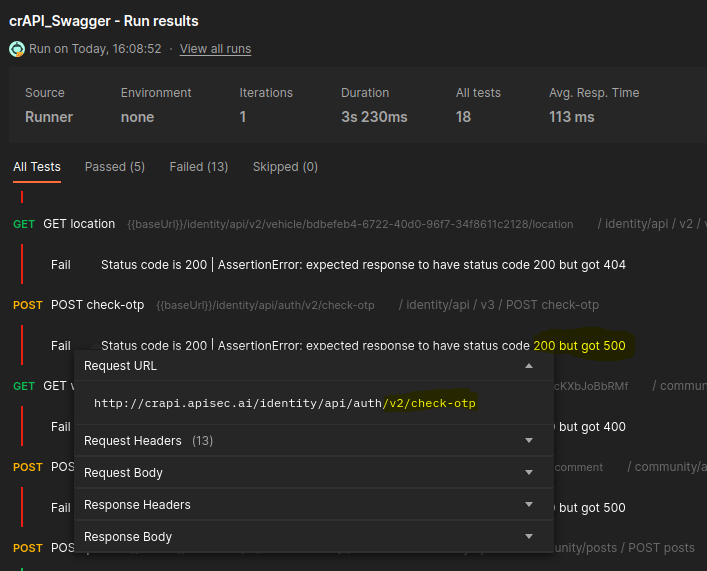
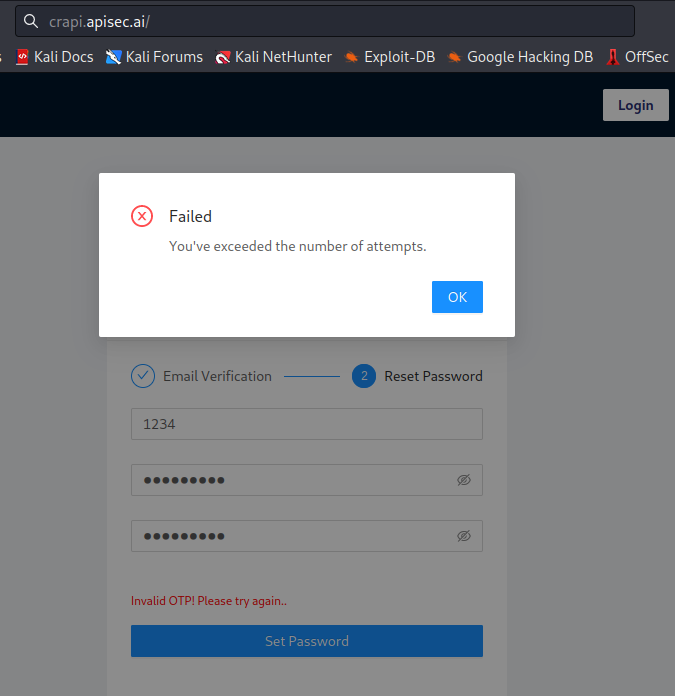
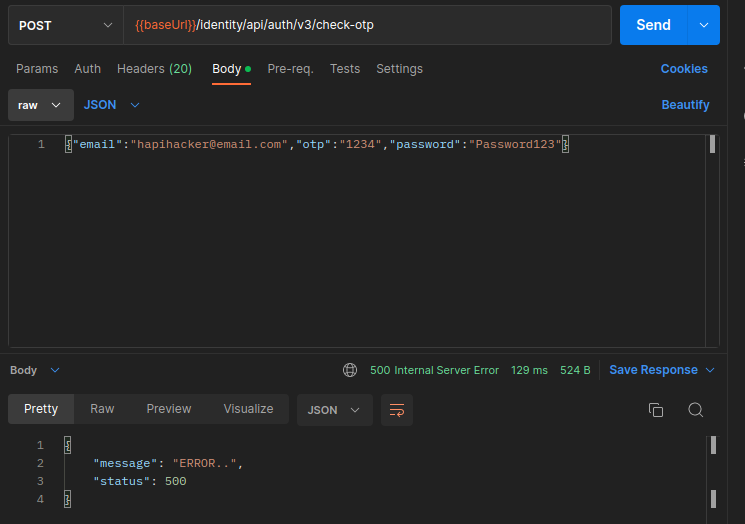
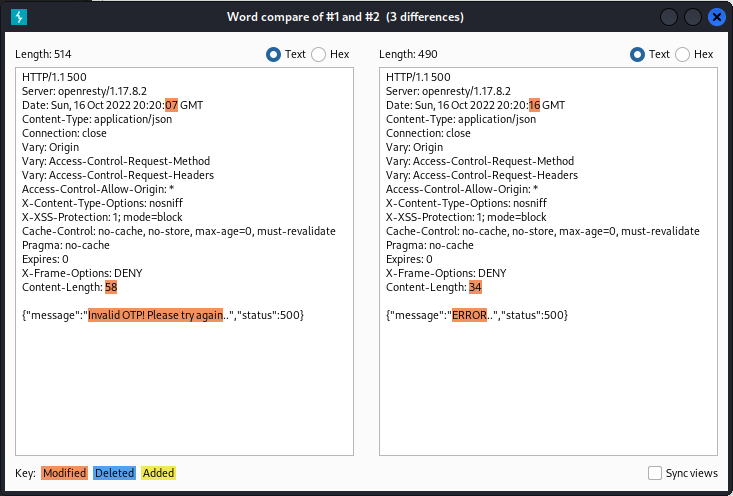
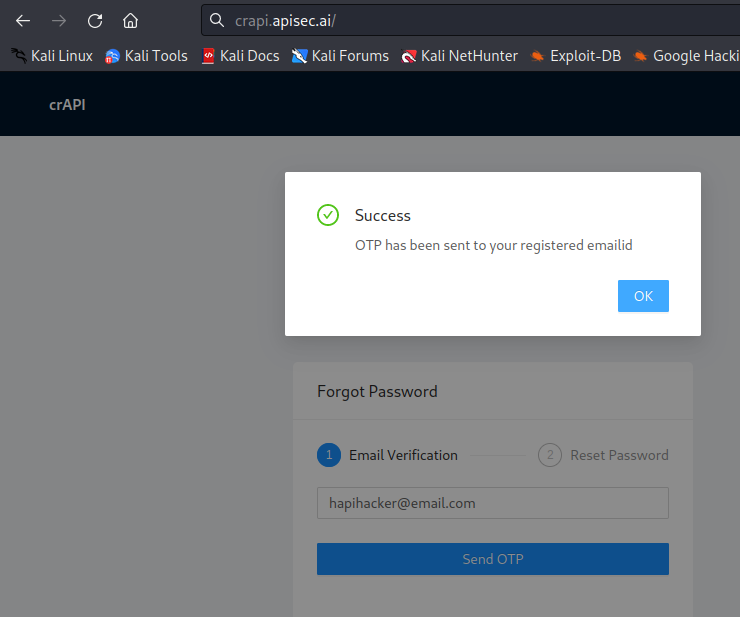
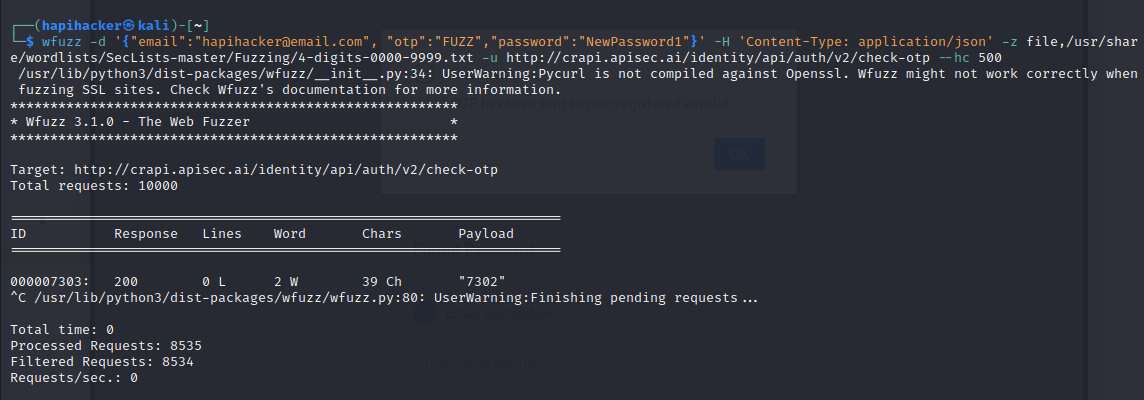
If you haven’t done so already, build a crAPI Postman collection and obtain a valid token. See the Setup module for instructions.

Finding Improper Assets Management Vulnerabilities

You can discover mass assignment vulnerabilities by finding interesting parameters in API documentation and then adding those parameters to requests. Look for parameters involved in user account properties, critical functions, and administrative actions. Intercepting API requests and responses could also reveal parameters worthy of testing. Additionally, you can guess parameters or fuzz them in API requests that accept user input. I recommend seeking out registration processes that allow you to create and/or edit account variables.

Improper Assets Management Testing

Now we can start by fuzzing requests across the entire API for the presence of other versions. Then we will pivot to focusing our testing based on our findings. When it comes to Improper Assets Management vulnerabilities, it is always a good idea to test from both unauthenticated and authenticated perspectives.

1. Understand the baseline versioning information of the API you are testing. Make sure to check out the path, parameters, and headers for any versioning information.  
   
2. To get better results from the Postman Collection Runner, we’ll configure a test using the Collection Editor. Select the crAPI collection options, choose Edit, and select the Tests tab. Add a test that will detect when a status code 200 is returned so that anything that does not result in a 200 Success response may stick out as anomalous. You can use the following test:  
   pm.test("Status code is 200", function () { pm.response.to.have.status(200); })  
   
3. Run an unauthenticated baseline scan of the crAPI collection with the Collection Runner. Make sure that "Save Responses" is checked as seen below.  
   
4. Review the results from your unauthenticated baseline scan to have an idea of how the API provider responds to requests using supported production versioning.  
   
5. Next, use "Find and Replace" to turn the collection's current versions into a variable. Make sure to do this for all versions, in the case of crAPI that means v2 and v3. Type the current version into "Find", update "Where" to the targeted collection, and update "Replace With" to a variable.  
    
6. Open Postman and navigate to the environmental variables (use the eye icon located at the top right of Postman as a shortcut).*Note, we are using environmental variables so that this test can be accessed and reused for other API collections.* Add a variable named "ver" to your Postman environment and set the initial value to "v1". Now you can update to test for various versioning-related paths such as v1, v2, v3, mobile, internal, test, and uat. As you come across different API versions expand this list of variables.  
   
7. Now that the environmental variable is set to v1 use the collection runner again and investigate the results. You can drill down into any of the requests by clicking on them. The "check-otp" request was getting a 500 response before and now it is 404. It is worth noting the difference, but when a resource does not exist, then this would actually be expected behaviour.  
   
8. If requests to paths that do not exist result in Success 200 responses, we’ll have to look out for other indicators to use to detect anomalies. Update the environmental variable to v2. Although most of the requests were already set to v2, it is worth testing because check-otp was previously set to v3.  
     
   Once again, run the collection runner with the new value set and review the results.  
     
   The /v2 request for check-otp is now receiving the same response as the original baseline request (to /v3). Since the request for /v1 received a *404 Not Found*, this response is really interesting. Since the request to /v2 is not a 404 and instead mirrors the response to /v3, this is a good indication that we have discovered an Improper Assets Management vulnerability. This is an interesting finding, but what is the full impact of this vulnerability?
9. Investigating the password reset request further will show that an HTTP 500 error is issued using the /v3 path because the application has a control that limits the number of times you can attempt to send the one-time passcode (OTP). Sending too many requests to /v3 will result in a different 500 response.  
   As seen from the browser:  
     
   As seen using Postman:  
     
     
   Sending the same request to /v2 also results in an HTTP 500 error, but the response is slightly larger. It may be worth viewing the two responses back in Burp Suite Comparer to see the spot differences. Notice how the response on the left has the message that indicates we guessed wrong but can try again. The request on the right indicates a new status that comes up after too many attempts have been made.   
     
     
   The /v2 password reset request responds with the body (left):  
   {"message":"Invalid OTP! Please try again..","status":500}  
     
   The /v3 password reset request responds with the body (right):  
   {"message":"ERROR..","status":500}  
     
   The impact of this vulnerability is that /v2 does not have a limitation on the number of times we can guess the OTP. With a four-digit OTP, we should be able to brute force the OTP within 10,000 requests.
10. To test this it is recommended that you use WFuzz, since Burp Suite CE will be throttled. First, make sure to issue a password reset request to your target email address. On the crAPI landing page select "Forgot Password?". Then enter a valid target email address and click "Send OTP".  
    
11. Now an OTP is issued and we should be able to brute force the code using WFuzz. By brute forcing this request, you should see the successful code that was used to change the target's password to whatever you would like. In the attack below, I update the password to "NewPassword1". Once you receive a successful response, you should be able to login with the target's email address and the password that you choose.   
    $ wfuzz -d '{"email":"hapihacker@email.com", "otp":"FUZZ","password":"NewPassword1"}' -H 'Content-Type: application/json' -z file,/usr/share/wordlists/SecLists-master/Fuzzing/4-digits-0000-9999.txt -u http://crapi.apisec.ai/identity/api/auth/v2/check-otp --hc 500  
      
    Within 10,000 requests, you’ll receive a 200 response indicating your victory. Congrats, on taking this Improper Assets Management vulnerability to the next level! Since we got sidetracked with this interesting finding during unauthenticated testing, I recommend returning to the crAPI collection and performing the same tests as an authenticated user.