**Assignment 2**

**OAuth authorization**

C.A. Hewagama - MS22909282

MSC. In cybersecurity

**Introduction**

Authentication and authorization are vital to the secure operation of any modern web application since it dictates who has what access to which resource. Without such measures web services would not be able to identify and allow legitimate users to access their services and would easily fall victim to cybercriminals and vandals.

The most straight forward method for allowing such access is user authentication via using a username and a password also known as direct authentication pattern[1]. There are some issues with standards username and password authentication methodology[2], but apart from that there are some problems with this approach when it comes to web applications.

If we look at web applications around 10 years ago, most of them were developed using server-side frameworks that rendered an HTML/JS/CSS web page sent to clients. When a client needs and update they would simply request another page from the server which is then piped back to them over HTTP[3, p. 27]. Such Traditional web applications were simple in nature and requires only a user login which can be fulfilled with http basic authentication pattern.

Today’s modern web applications have evolved from these more simpler web sites and have deviated away from the traditional monolithic web application paradigm. The arrival of ajax Ajax(asynchronous JavaScript and XML), supported web sites lead to more frequent use of HTTP and requests to be made from within the web page and serves to create single page web applications (e.g. Facebook, Gmail, twitter). Such modern applications nowadays typically operate as several application communicating with each other with the use of Representational State Transfer (REST) API. Some of these applications operate more like traditional desktop applications than web sites that came before them.

These web applications can deploy multiple microservices requiring a much more evolved approach since today’s complex microservices deployed by many web services require applications communicating with each other. A traditional user authentication would mean the user would have to hand over their credentials to the web service so they web service can login to another web service on their behalf to access a resource for them. Therefor a web application or a desktop application that uses web API to connect and access a resource using such credentials can create a number of different problems.

For example, let’s assume that there is a photo printing service that offers users to print and send them their photographs by a website. As an added feature the site could offer users the option to print the photos that reside in a customer’s google drive directly from the website for ease of use. However, the traditional login process quickly becomes problematic in such a scenario. Because the website doesn’t own the data it’s trying to access and it’s not trusted by google, so it wouldn’t allow the site to access one of its user’s private data. As a workaround the user could delegate the authentication and authorization process and hand over his login credentials to this photo printing service, but that creates a problem as well since there is no way for the user to limit the photo printing sites access only to his images. More importantly there is absolutely no guarantee that the website will not misuse the credentials, credentials wouldn’t fall on to someone else’s hands or the website will discard this sensitive information once the photos have been printed and sent out.

**OAuth: a way to authorize**

With more and more websites launching new services that will depend on functionality from other sites as shown above. These could be gaining location info to display information, getting address book details from a social media platform for a task or many others. Since users can’t simply hand over their credentials to every site or login every time to multiple sites when they visit them for some functionality or service a new way of granting access was required.

One such solution is OAuth which is discussed in this paper. OAuth is an open standard for authorization that’s seeing wide adoption today. OAuth allows decoupling of authentication from authorization and allows users to grant access to their private resources from one site (called a service provider in this case) to be accessed by another site (called consumer) without having to give away their credentials or revealing their identity[4]. So ultimately the goal of OAuth was to find a proper way to delegate authorization. OAuth is capable of supporting multiple use cases and devices/ platforms as well.

**What is OAuth?**

OAuth is an open standard for authorization of web services and was first introduced back in 2007. It was a result of the search for an open standard for API access delegation when no such standard existed at the time[4]. OAuth can be described as an authorization framework which essentially handles permissions. OAuth version 2.0 was introduced in 2013 and is not backwards compatible with OAuth 1.0[1].

Using OAuth provides the following advantages[5]

* utilizes HTTPS to communicate, therefore the data being sent between the communicators remains private.
* The tokenization mechanism allows for granting granular access to resources, which allows resource owners to share the resource and not worry about any sensitive data being accessed
* Users or resource owners can share data without divulging their identity
* Tokens can be revoked if not needed anymore
* Easier to implement and provides strong authentication

**OAuth scopes**

Scopes is a major component of OAuth which dictates what resources are given access and what’s not. Web application developers must design their application to ask for these permissions when they are being used. This consent must be retrieved from the resource owner (the web app user) so the application can get access to the resource and only that resource that has been allowed.

**Authentication and authorization process**

A picture containing Word

Description automatically generated

Figure 1 OAuth authorization flow [6]

OAuth process consists of the following parties:

* Resource: protected content that is desired and requested (e.g., photos on google drive)
* Resource owner: the person who is the owner of the resource, (e.g., user who owns the google account and owns the photos residing in their google drive)
* Resource server: google (google drive which holds the photos)
* Authentication server: google
* Client: the application which needs access to that protected resource on behalf of the resource owner (the photo printing service)

This authentication can be likened to access key cards commonly seen in business organizations or hotels that are used to allow or limit access to individuals based on privilege. Those who possess the key card would be able to gain access to areas that they key card is allowed. But to get the keycard they must first authenticate, for a hotel this would be at the front desk. Once the card is acquired, they don’t need to go back to the front desk anytime they want to access an area where they’re allowed.

It should be noted that for OAuth to function the client; ie the application that’s trying to access the protected resource on behalf of the resource owner, must first register with the resource server and get a client ID. This client ID identifies the client and establishes the trust between the authentication server and the client. There are two types of clients called confidential and public. Confidential clients can keep their client secret secure and are able to securely authenticate with authorization server. public clients can’t use registered client’s secret[7].

**OAuth token types**

OAuth uses different types of tokens for its functionality.

Access tokens: these are used to access resource server and has a short lifespan. These can be acquired by public clients as well.

Refresh token: these have much longer lifespans and are used to acquire new tokens and typically require client authentication to acquire. Refresh tokens can be revoked to prevent access if needed.

These two tokens are acquired from authorization server’s different endpoints. First the authorization endpoint grants the authorization grant when the resource owner gives authorization and consent to the web app. This authorization is then passed to token endpoint which then provides the access token and the refresh token. This access token is then used to access APIs.

It should be noted that authorization grant and tokens don’t need to be carried out over the same channel. If the OAuth is using token response type the token is sent through front channel which can be insecure since token can be intercepted by an attacker. Implicit grant follows this flow characteristic which is typically used for javascript apps that have no way of storing secrets. This flow is also known as 2 legged OAuth.

In response type code when auth mode is used the authentication server sends the auth token which is then exchanged again with resource server for an access token. This provides better security because to make the exchange the client must send their client ID and the client secret. So even if the auth token is intercepted by an attacker they can’t exchange it for an access token since they don’t possess the required client ID and the client secret. Furthermore, the process of authentication with authentication server and sending auth token is done in front channel. But auth token exchange with access token is direct server to server communication which is a back channel. This flow is also known as 3 legged OAuth.

The client credential flow is used in scenarios where server to server communication takes place. The application in this case is a confidential client and doesn’t act on behalf of any user. This flow is a back channel only flow. A legacy mode known as resource owner password flow exists in OAuth which resembles direct authentication pattern which typically doesn’t support refresh tokens. Assertion flow is another flow in OAuth which allows authorization to trust authorization grants from other external parties which allows SAML related services to be integrated with OAuth. Device flow is another flow in OAuth which doesn’t use a web browser and typically acts as a controller in devices like TVs.

**OAuth message flow in this scenario: login, file upload to google drive and viewing the file**

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidenceA picture containing graphical user interface

Description automatically generated

Diagram

Description automatically generatedDiagram

Description automatically generated

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

**OAuth in action**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

User authentication

Graphical user interface, application

Description automatically generated

**Login api call**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

**Drive API call**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Accessing uploaded file via link**

Graphical user interface, application

Description automatically generated

**Security considerations with OAuth usage**

* While OAuth is great at delegating authorization web security guidelines should be followed to ensure security.
* Using CSRF token when sending state parameter to ensure the integrity of the flow
* To ensure proper URI validations redirect URIs should be whitelisted
* Authorization grants and token requests should be bound to a client ID
* Ensuring that client ID is not leaked or put in a position where it can be accessed by unauthorized individuals

**Appendix - 1**

**References**

[1] M. Raible, “What the Heck is OAuth? | Okta Developer,” *Okta*, 2017. https://developer.okta.com/blog/2017/06/21/what-the-heck-is-oauth#why-oauth (accessed May 04, 2022).

[2] G. C. Kessler, “PASSWORDS — STRENGTHS AND WEAKNESSES,” 1996.

[3] H. Andrew, *Web Application Security - Exploitation and Countermeasures for Modern Web Applications*. O’Reilly, 2020.

[4] E. Hammer-Lahav, “Introduction — OAuth,” Nov. 05, 2007. https://oauth.net/about/introduction/ (accessed Apr. 26, 2022).

[5] B. Gorman, “Why Your Organization Should be Using OAUTH 2.0,” 2019. https://www.clowder.com/post/why-your-organization-should-be-using-oauth-2.0 (accessed May 04, 2022).

[6] M. Anicas, “An Introduction to OAuth 2 | DigitalOcean,” 2014. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 (accessed May 05, 2022).

[7] “Confidential and Public Clients - OAuth 2.0.” https://oauth.net/2/client-types/ (accessed May 05, 2022).

**Appendix - 2**

**Index.php**

<?php

// Include database configuration file

require\_once 'dbConfig.php';

$status\_msg = $valueErr = '';

$status = 'danger';

// If the form is submitted

if(isset($\_POST['submit'])){

// Form input field validation

if(empty($\_FILES["file"]["name"])){

$valueErr .= 'Please select a file to upload to google drive.<br/>';

}

// generate error if input is empty

if(empty($valueErr)){

$trgt\_dir = "uploads/";

$file\_name = basename($\_FILES["file"]["name"]);

$targetFilePath = $trgt\_dir . $file\_name;

// storing file to be uploaded

if(move\_uploaded\_file($\_FILES["file"]["tmp\_name"], $targetFilePath)){

// storing file ID and info to the database

//for reference

$sqlquery = "INSERT INTO drive\_files (file\_name,created) VALUES (?,NOW())";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("s", $db\_file\_name);

$db\_file\_name = $file\_name;

$insert = $statement->execute();

if($insert){

$file\_id = $statement->insert\_id;

// Store DB reference ID of file in SESSION

$\_SESSION['last\_file\_id'] = $file\_id;

header("Location: $OauthURL");

exit();

}else{

$status\_msg = 'connecitivty was not initialized successfully.';

}

}else{

$status\_msg = 'file upload failed. please contact system administrator';

}

}else{

$status\_msg = '<p>A file must be selected to be uploaded:</p>'.trim($valueErr, '<br/>');

}

}else{

$status\_msg = 'upload failed due an error!';

}

$\_SESSION['status\_response'] = array('status' => $status, 'status\_msg' => $status\_msg);

header("Location: index.php");

exit();

?>

**GDriveApi.class.php**

<?php

class GoogleDriveApi {

const OAUTH\_TOKEN = 'https://oauth2.googleapis.com/token';

const GDRIVE\_UPLOAD\_URI = 'https://www.googleapis.com/upload/drive/v3/files';

const GDRIVE\_FILE\_META = 'https://www.googleapis.com/drive/v3/files/';

function \_\_construct($params = array()) {

if (count($params) > 0){

$this->initialize($params);

}

}

function initialize($params = array()) {

if (count($params) > 0){

foreach ($params as $key => $val){

if (isset($this->$key)){

$this->$key = $val;

}

}

}

}

public function GetAccessToken($client\_id, $redirect\_uri, $client\_secret, $code) {

$curlPost = 'client\_id=' . $client\_id . '&redirect\_uri=' . $redirect\_uri . '&client\_secret=' . $client\_secret . '&code='. $code . '&grant\_type=authorization\_code';

$ch = curl\_init();

curl\_setopt($ch, CURLOPT\_URL, self::OAUTH\_TOKEN);

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($ch, CURLOPT\_POST, 1);

curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, FALSE);

curl\_setopt($ch, CURLOPT\_POSTFIELDS, $curlPost);

$data = json\_decode(curl\_exec($ch), true);

$http\_code = curl\_getinfo($ch,CURLINFO\_HTTP\_CODE);

if ($http\_code != 200) {

$error\_msg = 'Failed to receieve access token';

if (curl\_errno($ch)) {

$error\_msg = curl\_error($ch);

}

throw new Exception('Error '.$http\_code.': '.$error\_msg);

}

return $data;

}

public function UploadFileToDrive($access\_token, $file\_content, $mime\_type) {

$apiURL = self::GDRIVE\_UPLOAD\_URI . '?uploadType=media';

$ch = curl\_init();

curl\_setopt($ch, CURLOPT\_URL, $apiURL);

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($ch, CURLOPT\_POST, 1);

curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, FALSE);

curl\_setopt($ch, CURLOPT\_HTTPHEADER, array('Content-Type: '.$mime\_type, 'Authorization: Bearer '. $access\_token));

curl\_setopt($ch, CURLOPT\_POSTFIELDS, $file\_content);

$data = json\_decode(curl\_exec($ch), true);

$http\_code = curl\_getinfo($ch,CURLINFO\_HTTP\_CODE);

if ($http\_code != 200) {

$error\_msg = 'Failed to upload file to Google Drive';

if (curl\_errno($ch)) {

$error\_msg = curl\_error($ch);

}

throw new Exception('Error '.$http\_code.': '.$error\_msg);

}

return $data['id'];

}

public function listFiles($access\_token) {

$apiURL = self::GDRIVE\_UPLOAD\_URI;

$lst = curl\_init();

curl\_setopt($lst, CURLOPT\_URL, $apiURL);

curl\_setopt($lst, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($lst, CURLOPT\_POST, 1);

curl\_setopt($lst, CURLOPT\_SSL\_VERIFYPEER, FALSE);

$data = json\_decode(curl\_exec($lst), true);

$http\_code = curl\_getinfo($lst,CURLINFO\_HTTP\_CODE);

if ($http\_code != 200) {

$error\_msg = 'Failed to upload file to Google Drive';

if (curl\_errno($lst)) {

$error\_msg = curl\_error($lst);

}

throw new Exception('Error '.$http\_code.': '.$error\_msg);

}

return $data;

}

public function UpdateFileMeta($access\_token, $file\_id, $file\_meatadata) {

$googleURL = self::GDRIVE\_FILE\_META . $file\_id;

$ch = curl\_init();

curl\_setopt($ch, CURLOPT\_URL, $googleURL);

curl\_setopt($ch, CURLOPT\_RETURNTRANSFER, 1);

curl\_setopt($ch, CURLOPT\_POST, 1);

curl\_setopt($ch, CURLOPT\_SSL\_VERIFYPEER, FALSE);

curl\_setopt($ch, CURLOPT\_HTTPHEADER, array('Content-Type: application/json', 'Authorization: Bearer '. $access\_token));

curl\_setopt($ch, CURLOPT\_CUSTOMREQUEST, 'PATCH');

curl\_setopt($ch, CURLOPT\_POSTFIELDS, json\_encode($file\_meatadata));

$data = json\_decode(curl\_exec($ch), true);

$http\_return\_code = curl\_getinfo($ch,CURLINFO\_HTTP\_CODE);

if ($http\_return\_code != 200) {

$error\_msg = 'Failed to update file metadata';

if (curl\_errno($ch)) {

$error\_msg = curl\_error($ch);

}

throw new Exception('Error '.$http\_return\_code.': '.$error\_msg);

}

return $data;

}

}

?>

**Config.php**

<?php

// the database is used to store fileid of files that are uploaded

// Database connectivity configuration are defined here

define('DB\_HOST', 'localhost');

define('DB\_USERNAME', 'root');

define('DB\_PASSWORD', 'root');

define('DB\_NAME', 'gdrive');

// configuration for Google API connectivity

define('GOOGLE\_CLIENT\_ID', '');

define('GOOGLE\_CLIENT\_SECRET', '');

define('GOOGLE\_OAUTH\_SCOPE', 'https://www.googleapis.com/auth/drive');

define('REDIRECT\_URI', 'http://localhost/gdrive/google\_drive\_sync.php');

// starting session

if(!session\_id()) session\_start();

// OAuth URL for google

$OauthURL = 'https://accounts.google.com/o/oauth2/auth?scope=' . urlencode(GOOGLE\_OAUTH\_SCOPE) . '&redirect\_uri=' . REDIRECT\_URI . '&response\_type=code&client\_id=' . GOOGLE\_CLIENT\_ID . '&access\_type=online';

?>

**Dbconfig.php**

<?php

// Include the main configuration file

require\_once 'config.php';

// this creates a new db connection

$dbconn = new mysqli(DB\_HOST, DB\_USERNAME, DB\_PASSWORD, DB\_NAME);

// Checking if DB connection succeeded

if ($dbconn->connect\_error) {

die("Connection failed: " . $dbconn->connect\_error);

echo "database connectivity failed due to an error";

}

**google\_drive\_sync**

<?php

// Including Google drive api handling classes

include\_once 'GDriveApi.class.php';

//Including database conneciton configuration

require\_once 'dbConfig.php';

$statusMsg = '';

$status = 'danger';

//initialize calling google API

if(isset($\_GET['code'])){

$GoogleDriveApi = new GoogleDriveApi();

// check to see if there is previous file upload

// retrive reference ID of file from global session variable

$file\_id = $\_SESSION['last\_file\_id'];

//proceed if empty

if(!empty($file\_id)){

// Fetch file details from the database

$sqlquery = "SELECT \* FROM drive\_files WHERE id = ?";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("i", $db\_file\_id);

$db\_file\_id = $file\_id;

$statement->execute();

$result = $statement->get\_result();

$fileData = $result->fetch\_assoc();

if(!empty($fileData)){

$file\_name = $fileData['file\_name'];

$target\_file = 'uploads/'.$file\_name;

$file\_content = file\_get\_contents($target\_file);

$mime\_type = mime\_content\_type($target\_file);

// retrive the access token from google

if(!empty($\_SESSION['google\_access\_token'])){

$access\_token = $\_SESSION['google\_access\_token'];

}else{

$data = $GoogleDriveApi->GetAccessToken(GOOGLE\_CLIENT\_ID, REDIRECT\_URI, GOOGLE\_CLIENT\_SECRET, $\_GET['code']);

$access\_token = $data['access\_token'];

$\_SESSION['google\_access\_token'] = $access\_token;

}

//upload if the access token retrieval was successful

if(!empty($access\_token)){

try {

// Upload file to Google drive

$drive\_file\_id = $GoogleDriveApi->UploadFileToDrive($access\_token, $file\_content, $mime\_type);

if($drive\_file\_id){

$file\_meta = array(

'name' => basename($file\_name)

);

// Updating the metadata of the file in Google drive

$gdrive\_metadata = $GoogleDriveApi->UpdateFileMeta($access\_token, $drive\_file\_id, $file\_meta);

if($gdrive\_metadata){

// Update google drive file reference in the database

$sqlquery = "UPDATE drive\_files SET google\_drive\_file\_id=? WHERE id=?";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("si", $db\_gdrive\_file\_id, $db\_file\_id);

$db\_gdrive\_file\_id = $drive\_file\_id;

$db\_file\_id = $file\_id;

$update = $statement->execute();

unset($\_SESSION['last\_file\_id']);

unset($\_SESSION['google\_access\_token']);

$status = 'success';

$statusMsg .= '<p><a href="https://drive.google.com/open?id='.$gdrive\_metadata['id'].'" target="\_blank">'.$gdrive\_metadata['name'].'</a>';

$\_SESSION['pathmy'] = $statusMsg;

}

}

} catch(Exception $e) {

$statusMsg = $e->getMessage();

}

}else{

$statusMsg = 'Failed to fetch access token!';

}

}else{

$statusMsg = 'File data not found!';

}

}else{

$statusMsg = 'File reference not found!';

}

$\_SESSION['status\_response'] = array('status' => $status, 'status\_msg' => $statusMsg);

header("Location: index.php");

exit();

}

?>

Google\_drive\_sync.php

<?php

// Including Google drive api handling classes

include\_once 'GDriveApi.class.php';

//Including database conneciton configuration

require\_once 'dbConfig.php';

$statusMsg = '';

$status = 'danger';

//initialize calling google API

if(isset($\_GET['code'])){

$GoogleDriveApi = new GoogleDriveApi();

// check to see if there is previous file upload

// retrive reference ID of file from global session variable

$file\_id = $\_SESSION['last\_file\_id'];

//proceed if empty

if(!empty($file\_id)){

// Fetch file details from the database

$sqlquery = "SELECT \* FROM drive\_files WHERE id = ?";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("i", $db\_file\_id);

$db\_file\_id = $file\_id;

$statement->execute();

$result = $statement->get\_result();

$fileData = $result->fetch\_assoc();

if(!empty($fileData)){

$file\_name = $fileData['file\_name'];

$target\_file = 'uploads/'.$file\_name;

$file\_content = file\_get\_contents($target\_file);

$mime\_type = mime\_content\_type($target\_file);

// retrive the access token from google

if(!empty($\_SESSION['google\_access\_token'])){

$access\_token = $\_SESSION['google\_access\_token'];

}else{

$data = $GoogleDriveApi->GetAccessToken(GOOGLE\_CLIENT\_ID, REDIRECT\_URI, GOOGLE\_CLIENT\_SECRET, $\_GET['code']);

$access\_token = $data['access\_token'];

$\_SESSION['google\_access\_token'] = $access\_token;

}

//upload if the access token retrieval was successful

if(!empty($access\_token)){

try {

// Upload file to Google drive

$drive\_file\_id = $GoogleDriveApi->UploadFileToDrive($access\_token, $file\_content, $mime\_type);

if($drive\_file\_id){

$file\_meta = array(

'name' => basename($file\_name)

);

// Updating the metadata of the file in Google drive

$gdrive\_metadata = $GoogleDriveApi->UpdateFileMeta($access\_token, $drive\_file\_id, $file\_meta);

if($gdrive\_metadata){

// Update google drive file reference in the database

$sqlquery = "UPDATE drive\_files SET google\_drive\_file\_id=? WHERE id=?";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("si", $db\_gdrive\_file\_id, $db\_file\_id);

$db\_gdrive\_file\_id = $drive\_file\_id;

$db\_file\_id = $file\_id;

$update = $statement->execute();

unset($\_SESSION['last\_file\_id']);

unset($\_SESSION['google\_access\_token']);

$status = 'success';

$statusMsg .= '<p><a href="https://drive.google.com/open?id='.$gdrive\_metadata['id'].'" target="\_blank">'.$gdrive\_metadata['name'].'</a>';

$\_SESSION['pathmy'] = $statusMsg;

}

}

} catch(Exception $e) {

$statusMsg = $e->getMessage();

}

}else{

$statusMsg = 'Failed to fetch access token!';

}

}else{

$statusMsg = 'File data not found!';

}

}else{

$statusMsg = 'File reference not found!';

}

$\_SESSION['status\_response'] = array('status' => $status, 'status\_msg' => $statusMsg);

header("Location: index.php");

exit();

}

?>

**Upload.php**

<?php

// Include database configuration file

require\_once 'dbConfig.php';

$status\_msg = $valueErr = '';

$status = 'danger';

// If the form is submitted

if(isset($\_POST['submit'])){

// Form input field validation

if(empty($\_FILES["file"]["name"])){

$valueErr .= 'Please select a file to upload to google drive.<br/>';

}

// generate error if input is empty

if(empty($valueErr)){

$trgt\_dir = "uploads/";

$file\_name = basename($\_FILES["file"]["name"]);

$targetFilePath = $trgt\_dir . $file\_name;

// storing file to be uploaded

if(move\_uploaded\_file($\_FILES["file"]["tmp\_name"], $targetFilePath)){

// storing file ID and info to the database

//for reference

$sqlquery = "INSERT INTO drive\_files (file\_name,created) VALUES (?,NOW())";

$statement = $dbconn->prepare($sqlquery);

$statement->bind\_param("s", $db\_file\_name);

$db\_file\_name = $file\_name;

$insert = $statement->execute();

if($insert){

$file\_id = $statement->insert\_id;

// Store DB reference ID of file in SESSION

$\_SESSION['last\_file\_id'] = $file\_id;

header("Location: $OauthURL");

exit();

}else{

$status\_msg = 'connecitivty was not initialized successfully.';

}

}else{

$status\_msg = 'file upload failed. please contact system administrator';

}

}else{

$status\_msg = '<p>A file must be selected to be uploaded:</p>'.trim($valueErr, '<br/>');

}

}else{

$status\_msg = 'upload failed due an error!';

}

$\_SESSION['status\_response'] = array('status' => $status, 'status\_msg' => $status\_msg);

header("Location: index.php");

exit();

?>