Lab 4: Using OAuth for Authentication

1 Introduction

In this lab, you will write a web application that authenticates its users via Google and displays basic user account information as well as uploads the information to the Google Drive using Google APIs. Both authenticating users and calling Google APIs will be implemented using OAuth.

In class, you learned that one of the pitfalls of OAuth type systems is over-privilege. An over-privileged application increases the amount of damage that can occur if it is compromised. When an application has too few privileges (it is under-privileged), it does not work. However, if an application has too many privileges, there are no symptoms. The only defense against over-privilege is a good understanding of the privileges, what they are needed for and how they are acquired.

The other goal of this lab is to see and use token validation. You must call token validation at the appropriate points and ensure you interpret the results correctly.

The main goals of this lab are:

- Gain hands-on experience with a real system using OAuth
- Learn how to understand and navigate external documentation on a real security system
- Learn how to apply this principle of least privilege when requesting authorization in OAuth
- Learn how to perform token validation

Lab 4 should be done in groups of 2. It will be due at 11:59pm on Dec 7, 2015.

2 Background

It is increasingly common for web applications to authenticate their users via a popular social network such as Google or Facebook. This type of authentication not only reduces the number of passwords that a user needs to remember and the number of times that a user needs to login to use online services, but also offsets the challenges of implementing a secure authentication mechanism by reusing existing authentication services provided by popular social networks.

2.1 OAuth

OAuth is an open standard for secure authentication and authorization. It allows third-party web, mobile, and desktop applications to access protected data on behalf of a user from a web service. OAuth 2.0 is the latest version of OAuth. It is supported by many popular web services such as Google, Dropbox, Facebook, GitHub, and Windows Live. It has support from common programming languages including Java, Python, PHP, Javascript, .NET, and C++.

2.2 Google APIs

Google provides Google APIs to access its web services such as Google+, Google search, and Google book. Google APIs use OAuth 2.0 for authentication and authorization. An application needs to authenticate a user via Google and acquires authorization from Google before it can use Google APIs on behalf of the user.

3 Environment Setup

Following files contained in lab4-handout.tar will be given:

- bottle.py: bottle micro web framework
- run.sh: shell script to run the web server with command-line arguments
- google-api-python-client-1.4.2.tar.gz: Google API client library for python
- ece568app.py: skeleton web app
- ece568helper.py: containing helper methods for ece568app.py web app
- Templates:
    - views/
        - login.tpl
        - auth.tpl
        - drive.tpl

Note: You will submit only ece568app.py and auth.tpl

You will first need to create a new project on Google Developers Console (https://console.developers.google.com/). You will develop a web application for this project. After creating the new project, you should turn on Google+ API and Drive API under APIs & Auth for the project, so that your application can use those APIs to inquiry basic account information of a user and upload the data.

Then you need to create a new Client ID for your web application. Make sure you update the REDIRECT URIS properly whenever you need to. For example, if you run your web server on *localhost* with port number *12081* and want to allow redirect back to the page *auth.html*, then add and save the URI “http://localhost:12081/auth.html” to the entry for Authorized redirect URIs.

If you use localhost in your redirect URIs, then you will need to use browser on the same machine you are running the bottle web server. Otherwise, you can replace localhost with IP address of the machine your bottle web server is running on. You can type '**ifconfig**' and read the IP address from '**inet addr**' on a Linux machine. (or typing '**ipconfig**' and get it from "**IP address**" on a Windows machine) If you want to run the web server on another machine that is different from the machine you will run a web browser, then make sure your machine running the web browser can reach the machine running the web server. (You can try ping). Note that ECF machines are firewalled so you cannot access the webservice from another machine -- only localhost will work on ECF. When you submit your assignment, please assume that the web server is running on localhost.

To run the bottle web server, first make sure you have the provided bottle.py, micro web-framework for Python, on the current working directory. Then, try to run the given ece568app.py skeleton web app by typing '**./run.sh**' which will execute the shell script invoking 'python ece568app.py' with default arguments. If you see the error message saying the port is already in use, then adjust the port number accordingly (i.e. simply pick another port number and replace it in run.sh) to avoid confliction. To get more information, refer to the bottle tutorial: http://bottlepy.org/docs/dev/tutorial.html

To install the provided Google API Client Library for Python, untar the provided tar file, google-api-python-client-1.4.2.tar.gz, by typing '**tar xvf google-api-python-client-1.4.2.tar.gz**'. Then, you should go to the extracted directory and type '**python setup.py install --user**' to actually install the library locally.

4 Tasks

To complete this lab, you will learn how to use OAuth 2.0 API in Javascript as a client-side application and in Python as a web server application to authenticate users via Google and to access the user's basic account information using Google APIs. Also, you will need to use Google APIs to upload the retrieved user's account information to the Google Drive with the Google API Client Library. You will host your web application on the bottle micro web framework.

Overall, you will need to do the following tasks:

1. Complete the OAuth 2.0 client-side flow with the proper Redirect URIs
2. Perform token validation to ensure the token is authentic
3. Complete the OAuth 2.0 server-side flow with Google API Library
4. Retrieve the profile information and upload it to the Google Drive

Note: Only use Google API Client Library for Task 4.3 and 4.4, not for Task 4.1 and 4.2

4.1 Client-side flow of OAuth 2.0

Before you start working on this task, you should read Google's online document Using OAuth 2.0 to Access Google APIs for Client-side Applications (https://developers.google.com/accounts/docs/OAuth2UserAgent) to understand client-side flow of Google OAuth 2.0

First of all, you can run ece568app.py by using the provided run.sh. run.sh will use localhost as the default IP address and 12081 as the default port number. You can adjust these as described above in the Environment Setup section. run.sh will starts the bottle web server which will print out the address and the port on the console screen. For example, for localhost as IP address and 12081 as the port number, you will see the following outputs on the console screen:

```
-------------
$ ./run.sh
Bottle v0.13-dev server starting up (using WSGIRefServer())...
Listening on http://localhost:12081/
Hit Ctrl-C to quit.

127.0.0.1 - - [11/Nov/2015 23:31:30] "GET /login.html HTTP/1.1" 200 1073
-------------
```

run.sh not only specifies IP address and Port number, but also specifies your app’s Client ID you should have already created from the Google Developer Console. You will need to edit run.sh file and insert your Client ID before running the web server.

Then, you can visit to login.html. With the example above, you can open your browser and use the URL, http://localhost:12081/login.html, to browse the login.html page provided by ece568app.py.

For this task, your web application should be able to provide two web pages: the first web page authenticates users via Google, the second web page retrieves and displays user’s basic account information. The first web page is named login.html and the second web page is named auth.html. You should properly setup the google developer console page’s REDIRECT URIS setting for your web application.

When a user clicks on one of the first two Login URL links (those ones under the header “Display your Google profile” and “Display your Google profile with Email address”), she will be redirected to Google's login web page to authenticate herself to Google with her username and password. If the user successfully authenticates herself, Google will generate an access token, which will be used by your web application to call Google APIs, and redirect the user to your second web page, with the access token.

ece568helper.py contains code to pass arguments from ece568app.py to the corresponding template. ece568app.py also contains code that routes HTTP request to your web server to the proper template contained in views folder. For example, the function defined under "@route('login.html')" will handle HTTP request for login.html page. Then, the login() function will invoke the helper function get_login_html(...) defined in the ece568helper.py that takes five arguments such as addr, port, cid, scope1 and scope2. These arguments will be passed to the login.tpl and replace the placeholders where the corresponding variable names are enclosed by double curly brackets. (e.g. value contained in *addr* argument from ece568helper.py will replace *{{addr}}* in login.tpl)

Your task: Recall the principle of least privilege. Your job is to ensure that you format the request properly so that you retrieve the user's basic profile information only for the first link and the user's profile and e-mail address for the second link. Note that this can be done by controlling the scope1 and scope2 variables, which you will have to set correctly. Use the minimal scope so that the principle of least privilege is satisfied. Information on Google+ scopes can be found here: https://developers.google.com/+/web/api/rest/oauth#login-scopes The retrieved account information will be displayed under the "Profile Info:" on the auth.html page displayed in the browser. Using different scopes will cause different profile information to be displayed.

4.2 Access Token Validation

Once a user authenticates and grants permissions successfully via Google, the web browser will be redirected back to auth.html page with the access token as a part of the HTTP request to the auth.html page. auth.html page will get the access token, use it with Google+ API to retrieve the account information of the authenticated user and display it.

Retrieving the token from the request and invoking the Google+ API to retrieve profile information is already implemented in auth.tpl. However, the given code does not correctly validate the token. You will need to prevent an attack where an attacker is somehow able to steal or create a fake token and directly calls auth.html with the stolen or fake token. Your app should not accept such tokens and should refuse to allow the attacker to proceed. This attack is an example of a confused deputy problem (https://en.wikipedia.org/wiki/Confused_deputy_problem). In the context of OAuth 2.0, the confused deputy issue applies to implicit grant protocol flow when used for authentication, which is the case for this task using Google OAuth 2.0 for client-side applications. (refer to: http://stackoverflow.com/questions/17241771/how-and-why-is-google-oauth-token-validation-performed) This attack can be prevented by validating the input to the redirect page.

Insert your code in between "// ---- YOUR CODE BEGIN ----" and "// ---- YOUR CODE END ----" markers in the auth.tpl. You can make any change to the area between two markers, but do not modify any other part of the provided code in the auth.tpl

Your task: Protect your web app from the attack described above by implementing the token validation using javascript in auth.tpl. You should refer to "Validating the Token" section of the 'Using OAuth 2.0 for client-side applications' (https://developers.google.com/accounts/docs/OAuth2UserAgent). You can refer to getPeopleInfo() function in auth.tpl to learn how to invoke a Google API with the access token and how to handle the response. If the access token is invalid, then use:

- print_to_page("validity", validity_failed_msg);

to display string variable validity_failed_msg (i.e. "Validity check failed!") message to the element whose id is "validity". If the access token passes validation check you implemented, then use:

- print_to_page("validity", validity_succeeded_msg);

to display string variable validity_succeeded_msg (i.e. "Access Token is valid!") instead.

4.3 Server-side flow of OAuth 2.0

Before you start working on this task, you should read Google's online document Using OAuth 2.0 to Access Google APIs for Web Server Applications (https://developers.google.com/identity/protocols/OAuth2WebServer). It contains step-by-step instructions for you to follow to do this task. For this task, do not worry about the token validation unlike the previous task.

First, you will need to download client_secrets.json as described in the link and store it in the current working directory of ece568app.py. Then, you will implement driver() a handler for drive.html page request in ece568app.py. You will first go to login.html page and then click the third link under the header "Upload your Google profile to Google Drive". This will request drive.html page to your web app which will be handled by the drive(). drive() should redirect the user to the Google's OAuth 2.0 server, if the request does not contain authorization code. Then, the user will be authenticated via Google and redirect back to drive.html along with the authorization code if the requested permission via scope parameter is granted by the authenticated user. Unlike the client-flow, the web application will obtain the credentials (including access token) by exchanging the received authorization code.

Your Task: Fill in the variable SCOPE defined in ece568app.py properly with the minimal permissions your app needs to retrieve profile and upload the file to the Google Drive. (Scope for Drive API is described here: https://developers.google.com/drive/web/scopes) Once redirected back, the authorization code will be provided as the argument in the HTTP request if the user granted the permission. Get the authorization code string from the request (refer to http://bottlepy.org/docs/dev/tutorial.html and learn how to obtain query variables). Then, exchange the authorization code with the credentials (including access token). Finally, store the credentials to the ‘ece568app.out’ using the provided output_helper function in ece568helper.py by setting the key (i.e. 1st argument of the output_helper function) as ‘credentials’. (i.e. Simply use the following code snippet)

- ece568helper.output_helper('credentials', credentials)

Then, you can actually check the obtained credentials in the ‘ece568app.out’.

4.4 Upload the Profile information to the Google Drive

Now, let’s try to use Google API with the access token. We will use the provided Google API Client Library for Python. To install it, refer to the Environment Setup section above. In order to use Google API with the client library, you will first need to build the service object as described in https://developers.google.com/identity/protocols/OAuth2WebServer The example in the link used ‘drive’ for api_name and ‘v2’ for api_version as described in the Build the Service Object section here(https://developers.google.com/api-client-library/python/start/get_started) Refer to the link https://developers.google.com/api-client-library/python/apis/ in order to find the proper api_name and the api_version for the Google+ API.

Your Task: First, you should retrieve the user’s basic profile information only without email address. Build the service for Google+ API, and use a proper Person resource’s method (https://developers.google.com/+/web/api/rest/latest/people) to *get a person’s profile*. Retrieve the user’s basic profile information of the *authenticated user*. Once obtained the profile information as returned by the method ‘execute’, store it to the ‘profile.out’ by using output_helper again but this time using ‘profile’ as the key. (i.e. Simply use the following code snippet)

- ece568helper.output_helper('profile', people_document)

Finally, upload the profile file to the Google Drive by using a proper Files resource’s method (https://developers.google.com/drive/v2/reference/files) to *create a new file*. Use text/plain for the mime type so that you can verify that the file is properly loaded by reading the file content on the Google Drive via a Web Browser. Check your file is actually uploaded to Google Drive at https://drive.google.com/drive/my-drive

5 Deliverables

Please explain in at most 300 words, what you did to get the lab working. This explanation must be in the file explanations_lab4.txt. This file must also contain the names and student numbers of your group members prefixed by #. Do not prefix other lines by # as this would confuse the automated marking scripts.

```
#first1 last1, studentnum1, e-mail address1
#first2 last2, studentnum2, e-mail address2
```

It is very important that this information is correct as your mark will be assigned by student number, and the e-mail you give here will be how we will get the results of the lab back to you.

Submission is done using the ECF submit command:

```
submitece568f 4 ece568app.py auth.tpl explanations_lab4.txt
```