

## Topic 11: Authentication

**CITS3403 Agile Web Development** 

Reading: The Flask Mega-Tutorial Miguel Grinberg Chapter 5 Semester 1, 2022



- Security is a primary concern for anyone developing web applications.
- Data access must be controlled, passwords must be validated securely, and users just be able to trust the information presented to them.
- Complete security is very hard to achieve and beyond the scope of this unit, but basic authentication is relatively easy.
- An interesting case study of internet security is Anonymous's attack on HBGary:



arstechnica.com/tech-policy/2011/02/anonymous-speaks-the-inside-story-of-the-hbgary-hack/



# Web security makes use of the following basic concepts



- Public Key Encryption (eg RSA)
  - A public-private key is 2 functions *pub* and *priv* so that x = *priv(pub(x))* and given that you know *pub*, *priv* is hard to work out.
  - Public Key Encryption can be used for authentication. I can compute and publish *pub(x)* and only someone who knows *priv* can tell me what *x* is.
  - Public Key Encryption can be used for digital signatures. The pair (x, priv(x)) can be verified by anyone, but only created by some who knows priv.
  - Key distribution. A random key *x* can be generated and *pub(x)* can be sent to someone who knows *priv*. Then the pair knows *x*, but no body else does (even if they have been eaves dropping

## • Hashing (eg MD5)

- Secure hashing computes a large number from a stream of data, in such a way that it's very difficult to fabricate data with a certain hash.
- Different to hashing used for Hash tables etc.

#### Secure web session

- HTTP is stateless, so the server does not remember the client.
- For a secure session, every request needs to be authenticated... thankfully there are protocols to help here.
- SSL (secure sockets layer) wraps up the public key encryption process to enable a secure transaction.
- To use SSL we need to use the HTTPS protocol, which requires a signed certificate to allows users to trust the server.
- This prevents anyone from intercepting traffic from reading its contents.





#### **Cookies and Tokens**



- Web session security is managed through cookies and tokens.
- Cookies are packets of data stored in the browser.
  - Session cookies can record a users interaction with a site, persistent remain in your browser and allow sites to track your browsing habits.
  - Cookies consist of a name, a value and a set of attribute value pairs (e.g. expiration).
  - Cookies can be created and managed through javascript: document.cookie="trackme: false"; Pair Up!
  - Cookies are sent from the server to the browser:



#### CITS3403 group allocation tool, and flask sample project.

Pair-Up is a sample Flask application for CITS3403/CITS5505 students to register student groups for the project, and book demonstration times. To get started, register an account, and then enter your project team details

#### Registered project list

Find in page August All Match Case Whole Words								
🕞 🗘 Inspector 🕞 Console	Debugger	<pre>{} Style Editor</pre>	Performance	Memory ↑↓ Network     Network	k 🗄 Storage 🕇 Acce	ssibility		б] … ×
👻 🗄 Cache Storage	+ C'						Filter items	
http://drtnf.net.5000	Name	Domain	Path	Expires on	Last accessed on	Value	HttpOnly	sameSite
- E Cookies	session	drtnf.net	1	Session	Tue, 30 Apr 2019 07:10:33	.eJwlj0tqA0EMBe	true	Unset

- Authentication tokens allow you to store user • privaleges in JWT, (JSON web tokens), or other formats.
- These tokens, once granted are submitted • with web-requests to verify identities.



### Authentication and session management

- To manage users, a database can store user user data and password hashes. Unverified users are required to enter login details before a secure session commences.
- When we verify a users credentials against the database, the application can remember that all requests associated with that session come from the specified user.
- Therefore HTTP is stateless, but the application is not, and it can track the state (and authority) of every user that is logged in.
- Flask provides some basic tools for session management.





### **Elements of Web-security**



- Web security depends on trust. There are several elements to this:
  - 1. The web server needs to be confident that someone accessing data is authorised.
  - 2. The user needs to know that the site they are visiting is the one they intend to.
  - 3. Both the server and the client need to be confident that no one in the middle is accessing unauthorised data.
- 2 is typically handled by browsers, and 3 is achieved with https (week 11). In this lecture we'll focus on 1.

- To track a users identity we need to have them register so we can associate a user name with them.
- When someone uses an application a session is maintained via a variable held by the web-browser.
- When someone logs in they provide a password. This is salted and hashed to provide a digest which can compared to a hash in a database (keeping the password secure).
- Once the user is authenticated, they will be be served there requested pages, and their id will be a parameter of the requests.

#### **Adding authentication to Flask apps**



- In previous lectures we have looked at the MVC architecture, and linked in a simple database which contains a table of users.
- To add in authentication, we need every user to have a unique user id and a password, but we only want to store password hashes.
- The python package *werkzeug* (a part of flask) can handle the hashing.

```
>>> from werkzeug.security import generate password hash
>>> hash = generate password hash('foobar')
>>> hash
'pbkdf2:sha256:50000$vT9fkZM8$04dfa35c6476acf7e788alb5b3c35e217c78dc04539d295f011f01f18cd2175f
>>> from werkzeug.security import check password hash
>>> check password hash(hash, 'foobar')
True
>>> check password hash(hash, 'barfoo')
False
app/models.py: Password hashing and verification
from werkzeug.security import generate_password_hash, check password hash
# ...
class User(db.Model):
    # ...
    def set password(self, password):
       self.password_hash = generate_password_hash(password)
    def check password(self, password):
       return check password hash (self.password hash, password)
```

The password management can now be added to the User model, using werkzeug to generate and verify hashes.

### Flask Login Manager



- Flask-Login is a package that will automatically track secure sessions. It requires the User model to implement a number of methods and properties for checking if the user is authenticated, what their id is, etc.
- This functionality can be achieved by using the UserMixin, which implements those methods for you.
- As Flask-Login is agnostic to the database or ORM, we need to tell flask how to load a user.
- The decorator @login.user\_loader is for the method that maps an id to a user.

from app import db, login
<pre>from werkzeug.security import generate_password_hash, check_password_hash</pre>
from flask_login import UserMixin
에 있는 것은 가장에 있는 것은 것이 있는 것이다. 같은 것은 것이 같은 것이 있는 것이 같은 것이 있는 것이 있는 것이 있는 것이 같은 것이 있는 것이 있는 것이 있는 것이 있는 것이 같은 것이 있는 것이 같은 것이 같은 것이 같은 것이 있는 것이 있
#allows login to get student from database, given id
#will be stored as current_user?
@login.user_loader
def load_student(id):
<pre>return Student.query.get(int(id))</pre>
#student database will be prepopulated with
# student numbers, firstname, surname, CITS3403 boolean
#students can add/edit pin and project id
class <b>Student</b> (UserMixin, db.Model):
tablename='students'
<pre>id = db.Column(db.String(8), primary_key = True)#prepopulate</pre>
first_name = db.Column(db.String(64))#prepopulate
<pre>surname = db.Column(db.String(64))#prepopulate</pre>
<pre>prefered_name = db.Column(db.String(64))#defaults to first_name if empty</pre>
cits3403 = db.Column(db.Boolean)#prepopulate
<pre>password_hash = db.Column(db.String(128))#overkill to hash a four digit</pre>
pin, but included for learning.
<pre>project_id = db.Column(db.Integer, db.ForeignKey('projects.project_id'),</pre>
nullable=True) #assigned when project registered
# def registered(number):
# student = Student.query.filter_by(number = id.data).first()
<pre># if student.password_hash is not None:</pre>
<pre># raise validationError('Student is already registered')</pre>
# return True
det set password(selt password)*

def check\_password(self, password):
 return check\_password\_hash(self.password\_hash, password)

self.password hash = generate password hash(password)

## **Using Flask-Login**

- Flask-Login provides a methods login\_user, logout\_user and a variable current\_user (possibly anonymous) to manage sessions.
- login\_user will set current user to the from app import app, db
   specified user model.
- current\_user has a method is\_authenticated to check if they have provided login credentials.
- We also use a decorator @login\_required from Flask-Login to label the routes that require a login.
- Finally, in app/\_\_init\_\_.py, an instance of LoginManager is created, and the login\_view is set to the route login.

#### class StudentController(): def login(): form = LoginForm() if form.validate\_on\_submit(): #will return false for a get request student = Student.query.filter\_by(id=form.student\_number.data).first() if student is **None** or not student.check password(form.pin.data): flash('invalid username or data') return redirect(url for('login')) login\_user(student, remember=form.remember\_me.data) next\_page = request.args.get('next') if not next\_page or url\_parse(next\_page).netloc !='': next page = 'index' return redirect(url\_for(next\_page)) return render template('login.html',title="Sign in", form = form) def logout(): logout user() return redirect(url for('index')) from flask import render template, flash, redirect, url for from app import app, db from app.controllers import StudentController, ProjectController from flask import request from werkzeug.urls import url parse @app.route('/login', methods=['GET', 'POST']) def login(): if not current\_user.is\_authenticated: return StudentController.login() return redirect(url for('index')) @app.route('/logout') def logout(): return StudentController.logout() @app.route('/register', methods=['GET', 'POST']) def register(): return StudentController.register() @app.route('/new\_project', methods=['GET','POST']) @login\_required def new project(): if not current user.is authenticated: return redirect(url\_for('login')) 1 from flask import Flask return ProjectController.new project() 2 from config import Config 3 from flask sglalchemy import SOLAlchemy 4 from flask\_migrate import Migrate 5 from flask\_login import LoginManager 7 app = Flask( name ) 8 app.config.from object('config.TestingConfig') 9 db = SQLAlchemy(app) 10 migrate = Migrate(app, db) 11 login = LoginManager(app) 12 login.login view = 'login' from app import routes, models

app/\_\_init\_\_.py

#### **Updating the views**



- We can now use the current\_user variable in the templates we have built.
- The current\_user properties can be used to guard components of the web page that you only want logged in users to see, or to personalise the web



24	
25	Project Team
26	Project Description
27	Demo location
28	Demo time
29	{% if not current_user.is_anonymous %}
30	Action
31	{% endif %}
32	
33	{% for p in projects%}
34	Netro Million Million and Anna
35	{td>{{p['team']}}
36	{td>{{p['description']}}
37	{td>{{p['lab']}}
38	{td>{{p['time']}}
39	<pre>{% if not current_user.is_anonymous %}</pre>
40	
41	<pre>{% if p['project_id']== current_user.project_id %}</pre>
42	<a href='{{url_for("delete_project") }}'><span class="glyphicon glyphicon.garbage"><u>delete</u></span></a>
43	<a href='{{ url_for("edit_project") }}'><span class="glyphicon glyphicon.pencil"><u>edit</u></span></a>
44	{% endif %}
45	
46	{% endif %}
47	
48	{% endfor %}
49	
	③ drtnf.net:5000/index
	Home Enter Project details

```
<div class='container'>
16
         <div class='col-sm-4'>
18
         <a href='{{ url_for("index") }}'>Home</a>
19
         </div>
20
         {%if current_user.is_anonymous %}
         <div class='col-sm-4'></div>
         <div class='col-sm-4 text-right'>
23
           <a href='{{ url_for("login") }}'>Login</a><br/>
24
           <a href='{{ url_for("register") }}'>Register</a>
         </div>
26
         {% else %}
27
           {% if current user.project id == None %}
28
29
             <div class='col-sm-4 text-center'>
               <a href='{{ url_for("new_project") }}'>Enter Project details</a>
30
             </div>
31
           {% else %}
32
33
34
             <div class='col-sm-4 text-center'>
              <a href='{{ url_for("edit_project") }}'>Edit Project details</a>
             </div>
           {% endif %}
36
             <div class='col-sm-4 text-right'>
              <a href='{{ url_for("register") }}'>Update {{ current_user.prefered_name }}
  </a><br/>>
              <a href='{{ url_for("logout") }}'>Logout {{ current_user.prefered_name }}
   a>
             </div>
         {%endif %}
       </div>
       <hr>>
       {% with messages = get_flashed_messages() %}
       {% if messages %}
       {% for message in messages %}
         {{ message }} 
         {% endfor %}
       {% endif %}
       {% endwith %}
       <div class='container'>
       <h1>Pair Up!<h1>
           <h3>CITS3403 group allocation tool, and flask sample project.</h3>
           {% block content%}
           {% endblock %}
               ••• 🗵 🏠 🔍 Search
                                               Update Tim
                                               Logout Tim
```

#### **Alternative authentication methods**



- This type of authentication works well for web based sessions, but has a number of drawbacks.
  - It requires the application to track all user sessions which may not scale well.
  - HTTP requests are sent in plain text, and passwords should never be transmitted or stored in plain text.
  - The web is not only access through a browser, so how can we authenticate without sessions?
- JSON Web Tokens provide an alternative where a token is granted to a client, and the client must submit that token with every request.
- An added bonus is that using OAuth, external providers can check and grant tokens.





## OAuth and JWT



- OAuth (now OAuth2) was developped by Twitter to allow applications to authenticate and interact with Twitter, without requiring repeated logins.
- Oauth verifies a users identity and provides a JSON Web Token (JWT). This contains a header, a payload and a signature in compressed JSON.
- This header describes the encryption type, the payload typically provides some user identifier, an expiry and issuer information, and the signature is a secure hash, proving the token wasn't tampered with.
- You can configure flask to serve JWT tokens to clients, and verify those tokens, rather than checking session cookies with the flask\_oauth module.
- However, you can also have 3 parties like Google and Twitter, provide the tokens and do the validation.
- This saves you having to manage senesitive user data.



- The application or client requests authorization to the authorization server. This is performed through one of the different authorization flows. For example, a typical OpenID Connect compliant web application will go through the <u>/oauth/authorize</u> endpoint using the authorization code flow.
- When the authorization is granted, the authorization server returns an access token to the application.
- 3. The application uses the access token to access a protected resource (like an API).

### Authentication with passwords and tokens



- The web application used session based authentication, but there is no such session cookie for a REST API.
- Instead a token is granted to the user when they provide credentials, and requests augmented with that token user are assumed to come from the user.
- g is a context object that comes with each HTTP request

```
1 from flask import g
 2 from flask_httpauth import HTTPBasicAuth
 3 from app.models import Student
 4 from app.api.errors import error_response
 5 from flask httpauth import HTTPTokenAuth
 7 basic auth = HTTPBasicAuth()
 8 token auth = HTTPTokenAuth()
  9
 10 #password required for granting tokens
 11 @basic_auth.verify_password
12 def verify_password(student_number, pin):
     student = Student.query.get(student_number)
 13
     if student is None:
 14
 15
        return Flase
 16
     q.current user = student
     return student.check password(pin)
 17
 18
19 @basic_auth.error_handler
20 def basic_auth_error():
     return error_response(401)
 21
 22
 23 #token auth below
 24 @token_auth.verify_token
25 def verify_token(token):
     g.current_user = Student.check_token(token) if token else None
 26
27
     return g.current_user is not None
 28
29 @token_auth.error_handler
30 def token_auth_error():
 31
     return error_response(401)
app/api/auth.py
                                                            1,1
```

### Authentication with passwords and tokens



- The HTTPBasicAuth module is for verifying passwords in a request, which will grant a token.
- Then the HTTPTokenAuth can do token based authentication
- We need to update our User models so that the temporary token is kept in the database, as well as the password hash.
- When making changes to the models, remember to upgrade and migrate the changes to the database

```
project id = db.Column(db.Integer, db.ForeignKey('projects.project id'),nul
   lable=True)
     #token authetication for api
27
28
     token = db.Column(db.String(32), index=True, unique = True)
29
     token expiration=db.Column(db.DateTime)
30
31
     def set_password(self, password):
       self.password_hash = generate_password_hash(password)
32
33
     def check_password(self, password):
34
       return check password hash(self.password hash, password)
35
36
37
     ###Token support methods for api
 38
39
     def get_token(self, expires in=3600):
       now = datetime.utcnow()
41
       if self.token and self.token_expiration > now + timedelta(seconds=60):
42
          return self.token
43
       self.token = base64.b64encode(os.urandom(24)).decode('utf-8')
44
       self.token_expiration = now+timedelta(seconds=expires_in)
 45
       db.session.add(self)
46
       return self.token
47
48
     def revoke_token(self):
49
       self.token_expiration = datetime.utcnow() - timedelta(seconds=1)
50
51
     @staticmethod
52
     def check_token(token):
53
       student = Student.query.filter_by(token=token).first()
54
       if student is None or student.token_expiration < datetime.utcnow():
55
          return None
56
       return student
57
app/models.py
                                                                54,1
                                                                               13%
```

#### Interacting with the REST API

- To interact with a REST API, you can use a browser for GET requests, but others are not trivial
- The python package HTTPie can be used to send requests and receive responses.
- There are also graphical user interfaces, such as Postman for sending, receiving and testing in HTTP

(venv) \$ http GET http://localhost:5000/api/users/1
HTTP/1.0 200 OK
Content-Length: 457
Content-Type: application/json
Date: Mon, 27 Nov 2017 20:19:01 GMT
Server: Werkzeug/0.12.2 Python/3.6.3

```
" links": {
```

"avatar": "https://www.gravatar.com/avatar/993c...2724?d=identicon&s=128",
 "followed": "/api/users/1/followed",
 "followers": "/api/users/1/followers",
 "self": "/api/users/1"
},

```
"about_me": "Hello! I'm the author of the Flask Mega-Tutorial.",
"followed count": 0,
```

```
"follower_count": 1,
"id": 1,
"last seen": "2017-11-26T07:40:52.9428652",
```

```
"post_count": 10,
```

```
"username": "miguel"
```

(venv) \$ http --auth <username>:<password> POST http://localhost:5000/api/tokens
HTTP/1.0 200 OK
Content-Length: 50
Content-Type: application/json
Date: Mon, 27 Nov 2017 20:01:22 GMT
Server: Werkzeug/0.12.2 Python/3.6.3

(venv) \$ http POST http://localhost:5000/api/users username=alice password=dog \
 email=alice@example.com "about me=Hello, my name is Alice!"

"token": "pC1Nu9wwyNt8VCjltrWilFdFI276AcbS"

(venv) \$ http GET http://localhost:5000/api/users/1 \

"Authorization:Bearer pC1Nu9wwyNt8VCjltrWilFdFI276AcbS"