Universidad Nacional Autónoma de México
Facultad de Ciencias

Facultad de CIencias

Criptografía y Seguridad

Semestre: 2025-1

Equipo: Pingüicoders

Práctica 6:
SQLin

Arrieta Mancera Luis Sebastián - 318174116
Cruz Cruz Alan Josue - 319327133
García Ponce José Camilo - 319210536
Matute Cantón Sara Lorena - 319331622

Introducción:

En la práctica veremos cómo realizar un ataque de inyección de sql, al igual que cómo evitar que esto suceda. Esto con la intención de comprender lo sencillo que es realizar este tipo de ataque, y como nosotros al ser desarrolladores debemos asegurarnos de no dejar abierta esa puerta para un posible intruso. También es importante conocer este tipo de ataque ya que el atacante usa el conocimiento de la herramienta en nuestra contra

Desarrollo:

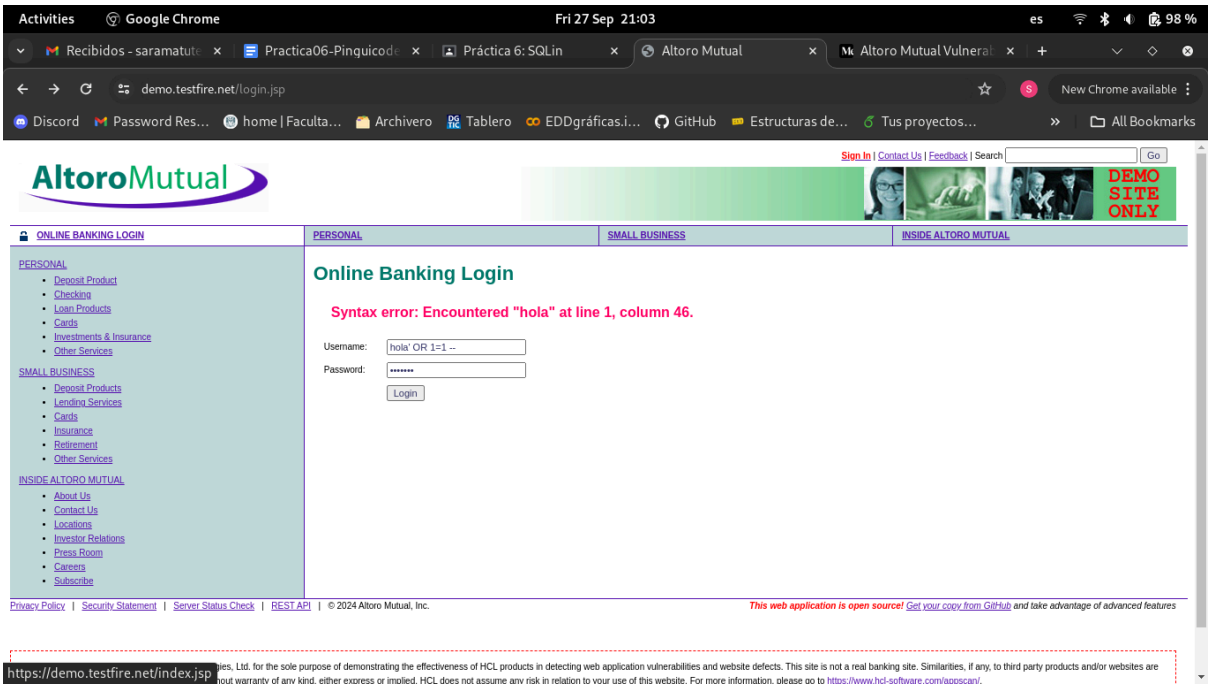
Parte 1:

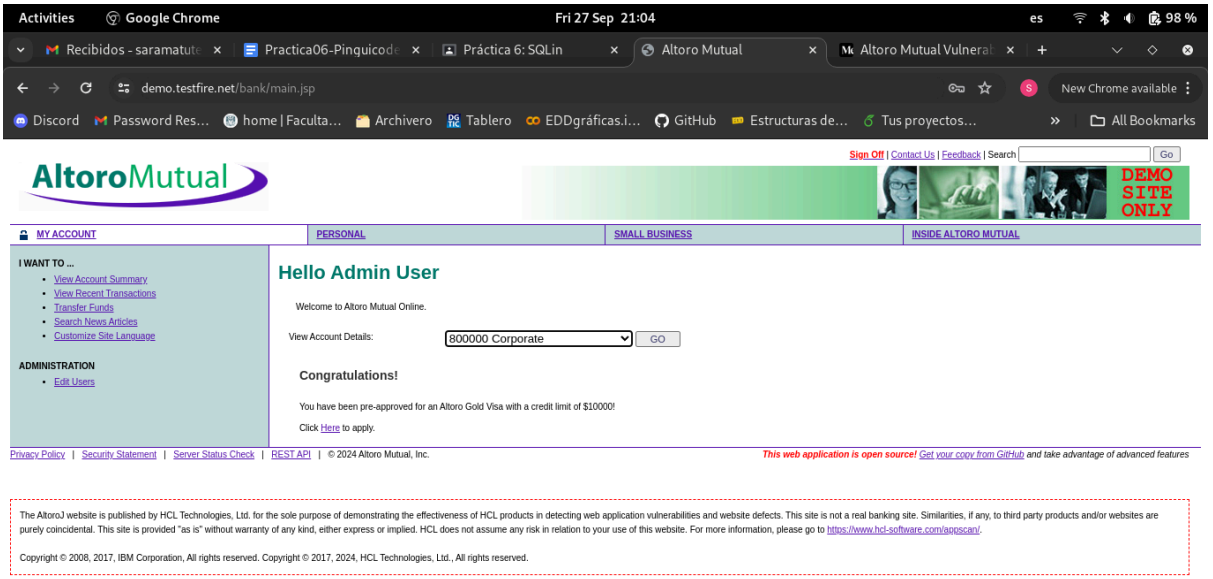
Para esta primera parte usamos el siguiente payload:

' OR 1=1 –

Para entrar con el siguiente “usuario”:

hola' OR 1=1 –





Parte 2:

1. ***[pdf]Inyección de SQL en áreas de texto.***

Forma para resolver esto, en cualquier query =text(f"algo"), comentarlo y usar placeholders en vez de f-strings

Cambio al validar usuario y contraseña

```python
def validate_user_and_password(username, password):
    # Cambio para evitar SQL Injection al iniciar sesion
    # En lugar de usar f-strings, usamos placeholders para los valores
    # Cambio para usar check_password_hash
    # En lugar de usar las contrasenias en texto plano, usamos hash
    session = db.session()
    #query = text(f"SELECT * FROM user WHERE username='{username}' AND password='{password}'")
    query = text("SELECT * FROM user WHERE username = :username")
    try:
        #cursor = session.execute(query).cursor
        result = session.execute(query, {'username': username})
        user = result.fetchone()
        pass
        if user:
            password1 = user.password
            resultado = check_password_hash(password1, password)
            if resultado:
                return True
    except:
        return False
    #return cursor.fetchone()
    return False
```

Cambio al obtener un usuario

```python
def get_user(username):
    # Cambio para evitar SQL Injection al iniciar sesion
    # En lugar de usar f-strings, usamos placeholders para los valores
    '''Returns the username as python object'''
    """
    session = db.session()
    query = text(f"SELECT * FROM user WHERE username = '{username}'")
    cursor = session.execute(query).cursor
    return cursor.fetchone()
    """
    session = db.session()
    query = text("SELECT * FROM user WHERE username = :username")
    result = session.execute(query, {'username': username})
    return result.fetchone()
```

2. ***[pdf] Vulnerabilidad en endpoints.***

Forma para resolver esto, en las páginas donde se pueda ver información de otros usuarios que no se tendría que ver, revisar si el usuario de la session es el correcto
Cambio al ver los posts de un usuario

```python
@home.route('/<user>/<access>', methods=['GET'])
def my_posts(user, access):
    '''See the public and private post of this user'''
    if access == 'private':
        access = False
    else:
        access = True
    # Cambio para evitar que un usuario vea los post privados de otro usuario
    current_user = session.get('user')
    if current_user != user:
        if access == False:
            flash('No puedes hacer esto >:(')
            return redirect(url_for('home.main'))
    posts = list(get_posts_by_access(user, access))[::-1]
    birthdays = get_birthdays(30)
    api_url = 'https://catfact.ninja/fact'
    response = requests.get(api_url).content.decode('utf8') #Trucos sobre trucos
    response = json.loads(response)
    today = datetime.datetime.now()
    return render_template('blog/home.html', posts = posts, birthdays = birthdays, today = today, response = response)
```

Cambio al ver un post en específico

```python
@home.route('/<int:id_post>')
@requires_login
def post(id_post):
    '''Gets a single post and performs operations on it'''
    post = get_post_by_id(id_post)
    # Cambio para evitar que un usuario vea los post privados de otro usuario
    current_user = session.get('user')
    if post.access == False:
        if current_user != post.author:
            if post.access == False:
                flash('No puedes hacer esto >:(')
                return redirect(url_for('home.main'))
    api_url = 'https://catfact.ninja/fact'
    response = requests.get(api_url).content.decode('utf8') #Trucos sobre trucos
    birthdays = get_birthdays(30)
    response = json.loads(response)
    today = datetime.datetime.now()
    return render_template('blog/post.html', post = post, id_post = id_post, birthdays = birthdays, response = response, today= datetime.datetime.now())
```

Cambio al ver el perfil de un usuario

```python
@profile.route('/<user>', methods=['GET', 'POST'])
@requires_login
def main(user):
    # Cambio para evitar que un usuario vea el perfil de otro usuario
    current_user = session.get('user')
    if current_user != user:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    '''Gets the user information and can update info when update'''
    user = get_user(user)
    if request.method == 'POST':
        first_name = request.form.get('first_name')
        second_name = request.form.get('second_name')
        birthday = request.form.get('birthday')
        old_password = request.form.get('old_password')
        new_password = request.form.get('new_password')
        # Cambio para usar generate_password_hash
        password_hash = generate_password_hash(new_password)
        # Cambio para actualizar los datos del usuario
        """
        user.first_name = first_name
        user.second_name = second_name
        user.birthday = birthday
        """
        # Cambio para actualizar la contrasenia
        """
        if old_password and new_password:
            if old_password == user.password:
                user.password = new_password
            else:
                flash('Las contraseñas no coinciden')
        """
        if old_password and new_password:
            if check_password_hash(user[1], old_password):
                # Cambio para actualizar los datos del usuario
                cambio = User.update(User, user[0], first_name, second_name, birthday, password_hash)
                if cambio == None:
                    flash('Error al actualizar los datos')
                else:
                    flash('Datos actualizados')
            else:
                flash('Contrasenia incorrecta')
        # Cambio para actualizar los datos del usuario
        #add(user)
    return render_template('blog/profile.html',
                           user = user)
```

Cambio al editar un post

```python
@profile.route('/update/<int:id_post>', methods=['GET', 'POST'])
@requires_login
def update(id_post):
    # Cambio implementacion para poder editar un post
    if request.method == 'POST':
        title = request.form.get('title')
        text = request.form.get('text').strip()
        img = request.files['img']
        access = 'access' in request.form
        if img:
            filename = secure_filename(img.filename)
            abs_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            img.save(abs_path)
            post_img = abs_path[5+8:]
            cambio = Post.update(Post, id_post, title, text, access, post_img)
        else:
            cambio = Post.update(Post, id_post, title, text, access, None)
        if cambio == None:
            flash('Error al actualizar el post')
        else:
            flash('Post actualizado')
        return redirect(url_for('home.main'))
    '''Updates a post by this user'''
    post = get_post_by_id(id_post)
    # Cambio para evitar que un usuario actualice un post de otro usuario
    current_user = session.get('user')
    if current_user != post.author:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    return render_template('blog/update.html', post = post)
```

Cambio al borrar un post

```python
@profile.route('/delete/<int:id_post>', methods=['GET', 'POST'])
@requires_login
def delete(id_post):
    '''Deletes a post by this user'''
    post = get_post_by_id(id_post)
    # Cambio para evitar que un usuario elimine un post de otro usuario
    current_user = session.get('user')
    if current_user != post.author:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    remove(post)
    # Cambio para redirigir a home.main en lugar de profile.main y agregar el mensaje de confirmación
    #return redirect(url_for('profile.main', user = post.author))
    flash('Post eliminado')
    return redirect(url_for('home.main'))
```

3. *[pdf]Vulnerabilidad en contraseñas.*

Forma para resolver esto, cuando se crea un usuario y cuando se inicia sesion, usar werkzeug.security para generate_password_hash y check_password_hash

Cambio al registrar un usuario

```python
@auth.route('/register', methods=['GET', 'POST'])
def register():
    # Cambio para usar generate_password_hash
    # En lugar de guardar la contrasenia en texto plano, guardamos el hash
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')
        password2 = generate_password_hash(password)
        first_name = request.form.get('first_name')
        second_name = request.form.get('second_name')
        birthday = request.form.get('birthday')

        user = get_user(username)
        if not user:
            user = User(username,
                        #password,
                        password2,
                        first_name,
                        second_name,
                        birthday)

            add(user)
            return redirect(url_for('home.main'))
        err = 'Username already exists!'
        flash(err)
    return render_template('auth/register.html')
```

Cambio al actualizar la contraseña de un usuario

```python
@profile.route('/<user>', methods=['GET', 'POST'])
@requires_login
def main(user):
    # Cambio para evitar que un usuario vea el perfil de otro usuario
    current_user = session.get('user')
    if current_user != user:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    '''Gets the user information and can update info when update'''
    user = get_user(user)
    if request.method == 'POST':
        first_name = request.form.get('first_name')
        second_name = request.form.get('second_name')
        birthday = request.form.get('birthday')
        old_password = request.form.get('old_password')
        new_password = request.form.get('new_password')
        # Cambio para usar generate_password_hash
        password_hash = generate_password_hash(new_password)
        # Cambio para actualizar los datos del usuario
        """
        user.first_name = first_name
        user.second_name = second_name
        user.birthday = birthday
        """
        # Cambio para actualizar la contrasenia
        """
        if old_password and new_password:
            if old_password == user.password:
                user.password = new_password
            else:
                flash('Las contraseñas no coinciden')
        """
        if old_password and new_password:
            if check_password_hash(user[1], old_password):
                # Cambio para actualizar los datos del usuario
                cambio = User.update(User, user[0], first_name, second_name, birthday, password_hash)
                if cambio == None:
                    flash('Error al actualizar los datos')
                else:
                    flash('Datos actualizados')
            else:
                flash('Contrasenia incorrecta')
        # Cambio para actualizar los datos del usuario
        #add(user)
    return render_template('blog/profile.html',
                           user = user)
```

Cambio al validar usuario y contraseña

```python
def validate_user_and_password(username, password):
    # Cambio para evitar SQL Injection al iniciar sesion
    # En lugar de usar f-strings, usamos placeholders para los valores
    # Cambio para usar check_password_hash
    # En lugar de usar las contrasenias en texto plano, usamos hash
    session = db.session()
    #query = text(f"SELECT * FROM user WHERE username='{username}' AND password='{password}'")
    query = text("SELECT * FROM user WHERE username = :username")
    try:
        #cursor = session.execute(query).cursor
        result = session.execute(query, {'username': username})
        user = result.fetchone()
        pass
        if user:
            password1 = user.password
            resultado = check_password_hash(password1, password)
            if resultado:
                return True
    except:
        return False
    #return cursor.fetchone()
    return False
```

Esta solución hace que los usuarios originales de la base de datos no puedan iniciar sesión, a menos que se actualice su contraseña

4. ***No permite modificar datos del usuario***

Forma para resolver esto, agregar una función para poder editar los datos del usuario
Método para la actualización

```python
# Cambio para actualizar los datos del usuario
def update(self, username, first_name, second_name, birthday, password):
    user = User.query.filter_by(username=username).first()
    if user:
        user.first_name = first_name
        user.second_name = second_name
        user.birthday = birthday
        user.password = password
        db.session.commit()
        return user
    else:
        return None
```

Cambio en el perfil del usuario

```python
@profile.route('/<user>', methods=['GET', 'POST'])
@requires_login
def main(user):
    # Cambio para evitar que un usuario vea el perfil de otro usuario
    current_user = session.get('user')
    if current_user != user:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    '''Gets the user information and can update info when update'''
    user = get_user(user)
    if request.method == 'POST':
        first_name = request.form.get('first_name')
        second_name = request.form.get('second_name')
        birthday = request.form.get('birthday')
        old_password = request.form.get('old_password')
        new_password = request.form.get('new_password')
        # Cambio para usar generate_password_hash
        password_hash = generate_password_hash(new_password)
        # Cambio para actualizar los datos del usuario
        """
        user.first_name = first_name
        user.second_name = second_name
        user.birthday = birthday
        """
        # Cambio para actualizar la contrasenia
        """
        if old_password and new_password:
            if old_password == user.password:
                user.password = new_password
            else:
                flash('Las contraseñas no coinciden')
        """
        if old_password and new_password:
            if check_password_hash(user[1], old_password):
                # Cambio para actualizar los datos del usuario
                cambio = User.update(User, user[0], first_name, second_name, birthday, password_hash)
                if cambio == None:
                    flash('Error al actualizar los datos')
                else:
                    flash('Datos actualizados')
            else:
                flash('Contrasenia incorrecta')
        # Cambio para actualizar los datos del usuario
        #add(user)
    return render_template('blog/profile.html',
                           user = user)
```

5. ***En el perfil se muestra la contraseña en vez del primer nombre y también se muestra el primer nombre en lugar de segundo nombre***

Forma para resolver esto, modificar el html para que lea los valores correctos

```
{% extends 'blog/home.html' %} {% block title %}
<title>Perfil</title>
{% endblock %} {% block content %}
<h2>Profile</h2>

<form method="POST">
  {{ macro.label_input('Username', 'username', placeholder=user[0], disabled='')
  }}
  <!--
  {{ macro.label_input('First Name:', 'first_name', value=user[1]) }}
  {{ macro.label_input('Second Name:', 'second_name', value=user[2]) }}
-->
  <!-- Cambio para no mostrar la contrasenia en el perfil -->
  {{ macro.label_input('First Name:', 'first_name', value=user[2]) }} {{
  macro.label_input('Second Name:', 'second_name', value=user[3]) }} {{
  macro.label_input('Birthday:', 'birthday', type='date', value=user[4] ) }} {{
  macro.label_input('Old password', 'old_password', type='password') }} {{
  macro.label_input('New password', 'new_password', type='password') }} {{
  macro.label_input(value='Update', type='submit') }}
</form>

{% endblock %}
```

6. ***Para todos los archivos que usan flash() falta importar flash from flask***

Forma para resolver esto, importar flash

```
# Cambio import de flash from flask
from flask import (jsonify, render_template, Blueprint, g, redirect, request, session, url_for, flash)
```

```
# Cambio import de flash from flask
from flask import (render_template, Blueprint, g, redirect, request, session, url_for, flash)
```

7. ***No permite modificar posts***

Forma para resolver esto, agregar una función para poder editar los datos del post
Método para la actualización

```
# Cambio para actualizar los datos del post
def update(self, id, title, text, access, img):
    post = Post.query.get(id)
    if post:
        post.title = title
        post.text = text
        post.access = access
        if img != None:
            post.img = img
        db.session.commit()
        return post
    else:
        return None
```

Cambio en el update del post

```
@profile.route('/update/<int:id_post>', methods=['GET', 'POST'])
@requires_login
def update(id_post):
    # Cambio implementacion para poder editar un post
    if request.method == 'POST':
        title = request.form.get('title')
        text = request.form.get('text').strip()
        img = request.files['img']
        access = 'access' in request.form
        if img:
            filename = secure_filename(img.filename)
            abs_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
            img.save(abs_path)
            post_img = abs_path[5+8:]
            cambio = Post.update(Post, id_post, title, text, access, post_img)
        else:
            cambio = Post.update(Post, id_post, title, text, access, None)
        if cambio == None:
            flash('Error al actualizar el post')
        else:
            flash('Post actualizado')
        return redirect(url_for('home.main'))
    '''Updates a post by this user'''
    post = get_post_by_id(id_post)
    # Cambio para evitar que un usuario actualice un post de otro usuario
    current_user = session.get('user')
    if current_user != post.author:
        flash('No puedes hacer esto >:(')
        return redirect(url_for('home.main'))
    return render_template('blog/update.html', post = post)
```

Cambio en el html del update

```
{% extends 'blog/home.html' %} {% block title %}
<title>Update post</title>
{% endblock %} {% block script %}
<!-- Cambio para poder editar el post -->
<script
  type="text/javascript"
  src="{{ url_for('static', filename='js/script.js') }}"
></script>
{% endblock %} {% block content %}
<div>
  <h3>Update post</h3>
</div>

<form enctype="multipart/form-data" method="POST">
  {{ macro.label_input('Title:', 'title', required='', value=post.title) }} {{
  macro.label_input('Description:', 'text', type='textarea', required='',
  value=post.text) }} {{ macro.label_input('Image:', 'img', type='file',
  id='file') }} {{ macro.label_input('Public:', 'access', type='checkbox',
  checked=post.access) }} {{ macro.label_input('', 'submit', value='Update',
  type='submit') }}
</form>
{% endblock %}
```

Preguntas

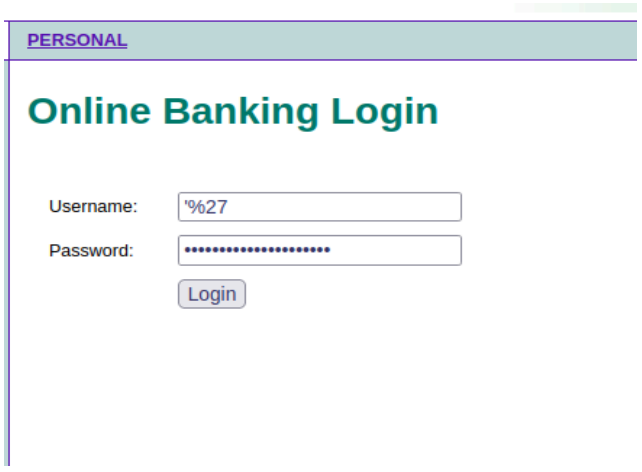
1. Investiga al menos otros 5 payloads para la inyección de queries en SQL y cómo es que funcionan.

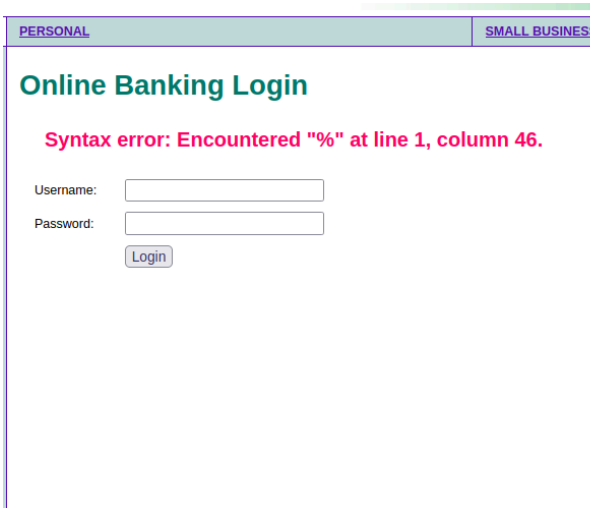
-Detectar si una página es vulnerable a inyecciones SQL

En el formulario que tenga una página web podemos usar varios payloads para comprobar si es vulnerable a inyecciones SQL. Algunos de los payloads que podemos usar son:

‘ %27 “ %22 # %23 ; %3B) %29

Estos payloads provocarán un error de sintaxis para la base de datos, como en Altoro





Si se genera un error que podamos ver, en este caso las letras rojas indicando un error de sintaxis, eso quiere decir que el sitio es vulnerable. Esto porque se introduce en el username de verificación caracteres que en las consultas SQL generan errores, por ejemplo cuando ponemos ‘ en la consulta se ve como:

SELECT * FROM logins WHERE username='' AND password = 'algo';

Lo cuál genera un error de sintaxis para SQL porque las comillas simples son como límites para las cadenas y siempre van en pares.

-Consultar el número de columnas

En el formulario ponemos el payload:

' ORDER BY n--

Siendo n el número de columnas. ORDER BY lo que hace es ordenar los resultados de la columna que consultamos, entonces podemos empezar con *order by 1,* luego ORDER BY *2 ,* ORDER BY *3, ..* y así sucesivamente hasta que nos marque un error, si por ejemplo nos marca error cuando ponemos order by 5 quiere decir que tiene 4 columnas.

El “--” es para comentar el resto de la consulta y que no se tome en cuenta, es algo que veremos en los demás payloads así que es bueno aclararlo. Se usa para que los ataques no se invaliden por alguna condición extra.

-Ataques de UNION

Los ataques UNION es un tipo de inyección SQL, hay varios tipos y se usan para obtener información de las tablas de la base de datos. El comando UNION en SQL sirve para ejecutar una consulta más o varias. Algunas veces sucede que la cantidad de consultas SQL (en el payload) es diferente a la de columnas por eso se usa *SELECT* para ejecutar otras consultas más y después agregar los resultados en la primera consulta, por ejemplo (el siguiente ejemplo es citado directamente del artículo Inyecciones SQL por Dalton):

SELECT a, b FORM table1 UNION SELECT c, d FROM table2

La consulta devuelve solo un conjunto de resultados con dos columnas, que contiene valores de columnas a y b en table1 y c y d en table2.

Bueno ahora sí. Cuando queremos consultar el número de columnas con UNION usamos SELECT junto con el número de columnas que creemos que hay hasta que falle porque es ahí cuando habremos superado el número de columnas y con eso reducir la cantidad de columnas que hay, por ejemplo digamos que hay cuatro columnas por lo que el payload es:

' UNION SELECT 1,2,3,4-- -

Nota: En otras fuentes ponen la consulta como *' UNION SELECT NULL,NULL,NULL,NULL--*

Y con el número de columnas en la base de datos se pueden usar otras consultas de SQL para obtener información relevante.

De aquí se derivan varios otros payloads.

-Encontrar el tipo de dato que se usa en una columna

Para este payload se deben de conocer la cantidad de columnas, en los payloads se inserta un tipo de dato y experimentar hasta que salga un error, por ejemplo:

' UNION SELECT 'hola',NULL,NULL--
' UNION SELECT NULL,'hola',NULL--
' UNION SELECT NULL,NULL,'hola'--

Se inserta en cada columna la cadena '*hola'* y si el tipo de dato no coincide con la columna arroja un error diciendo que no se ha podido transformar el dato introducido al tipo que tiene la columna, por ejemplo se podría ver algo así como:

Conversion failed when converting the varchar value 'hola' to data type int.

Indicando que en una de las columnas el tipo de dato que se usa son enteros. En algunos casos si resulta que el tipo de dato introducido coincide con el tipo de dato de las columnas entonces devolverá los resultados del SELECT.

-Consultar la versión de la base de datos

Cuando se busca vulnerar una base de datos a veces es necesario recopilar información respecto al software que se utiliza, una vez más usamos el comando UNION como herramienta. Dependiendo del software variará en las consultas que siguen después de UNION, el payload es:

' UNION SELECT *[consulta]* –

No es muy complicado dar con la consulta correcta ya que podemos probar las consultas de los programas más usados:

MySQL, Microsoft SQL Server usan *@@version*
Oracle usa ** FROM v$version*
PostgreSQL usa version()

Por ejemplo " ' *UNION SELECT @@version* – " devuelve la versión si se utiliza MySQL.

2. ¿Qué es el soft delete y el hard delete y en qué caso es bueno usar cada uno?

Ambas son formas en las que se eliminan registros de una base de datos o en otros sistemas que almacenen cualquier tipo de información.

El soft delete es la técnica en la que no borramos un registro de la base de datos si no que lo desactivamos, como si comentáramos código aunque en bases de datos se suele agregar una columna para indicar si se borró como "is_deleted" que toma valores true o false. Es bueno usarlo cuando no ocupamos un registro de forma temporal pero puede que en un futuro tengamos que volver a requerir del registro

Un ejemplo sería cuando borramos un archivo de nuestra carpeta en el sistema operativo, normalmente el archivo no se borra por completo si no que se va a la papelera donde es posible recuperarlo si cometimos un error o algo por el estilo.

Hard delete es borrar el registro de la base de datos de forma permanente. A menos de que tengamos un respaldo del registro, ya no podremos recuperarlo una vez que lo hayamos borrado con hard delete. Es bueno cuando queremos deshacernos de registros de los cuáles estamos muy seguros que no volveremos a usar, ya sea por espacio o porque no aporta al sistema.
Siguiendo el ejemplo anterior, si limpiamos la papelera todos los archivos que estaban ahí se borrarán por completo y no podremos recuperarlo(a menos que tengamos un respaldo).

3. Traducido a español, ¿qué nos dice la siguiente consulta?
SELECT user, nip, amount FROM user.credit_card WHERE user='pichulanda' and status=1;

Parece que es de una tabla que contiene datos de las tarjetas de crédito.
Lo que se busca consultar es el nip, mount (monto de tarjeta), user (nombre de usuario al parecer) de la tarjeta de crédito (la tabla) del usuario que se llama "pichulanda" y que además tiene de status 1, posiblemente quiere comprobar que el usuario tenga una tarjeta de crédito activa.

4. Si queremos sacar toda la información de todas las tarjetas de crédito de todos los usuarios de la anterior consulta, ¿cómo debería quedar la consulta, sustituyendo un payload?

La consulta podria ser asi SELECT * FROM user.credit_card; y usando el payload "' OR '1'='1' --", también podríamos de la siguiente manera SELECT user, nip, amount FROM user.credit_card WHERE user='' OR '1'='1' --' AND status=1;

5. ¿Qué hace el siguiente evento canónico? ¿Qué le falta?
DELETE * FROM user;

El comando indica que borrará todos los registros de las columnas de la tabla "user", si lo que queremos es solamente eliminar los datos que cumplen con cierto parámetro entonces agregamos "WHERE" seguida de la condición,por ejemplo "DELETE FROM user WHERE status = 0 " si lo que queremos es eliminar los registros de los usuarios que no tienen su tarjeta activa (siguiendo los ejemplos anteriores), y así ya no borramos todos los datos registrados en la tabla.

Conclusiones:

Lo que se ha usado en esta práctica es usado por profesionales pentesters, aunque ellos suelen usar técnicas más complejas que solo consultas básicas de inyecciones SQL. Tener los conocimientos básicos respecto a este tipo de vulnerabilidades

ayuda a crear sistemas, como lo pueden ser una página web, más seguras porque el simple hecho de que puedan usar un payload que les ayude a saber si la página es vulnerable puede ser nuestro final.
Aprendimos qué tan poderosos podemos ser ante las páginas que no están seguras con unas simples consultas lo que nos hizo reflexionar en que al hacer backend es importante darle relevancia a estos aspectos de seguridad. Además sirvió como repaso de los temas que vimos en Fundamentos de Bases de Datos y en Ingeniería del Software.
La segunda parte de la práctica creemos que es la más interesante porque se siente como un trabajo real, basándonos en las anécdotas de ayudantes y otros conocidos que trabajan en la industria, por el hecho de “parchear” el código de un sistema que tiene varios errores para que funcione de manera óptima

Referencias:

-Bisht, S. S. (2023, 19 octubre). Understanding Soft Delete and Hard Delete in Software Development: Best Practices and Importance. *Medium*. https://surajsinghbisht054.medium.com/understanding-soft-delete-and-hard-delete-in-software-development-best-practices-and-importance-539a935d71b5

-*Soft and Hard Delete: everything you need to know | Óscar Martínez*. (2024, 21 febrero). https://oscmarb.com/blog/soft-delete-and-hard-delete-everything-you-need-to-know/

-*W3Schools.com*. (s. f.). https://www.w3schools.com/sql/sql_delete.asp

-Dalton. (2023, 12 mayo). Inyecciones SQL - Dalton - Medium. *Medium*. https://dalthon.medium.com/inyecciones-sql-505eef46dd88
-Bytemind. (2023b, febrero 5). *Sql Injection tipos ejemplos y contramedidas*. Byte Mind. https://byte-mind.net/sql-injection-tipos-ejemplos-y-contramedidas/