Vulnerabilidades en el tratamiento de los datos de entrada

Una vulnerabilidad es algo que no está bien implementado o implementado de forma débil, de forma que esta debilidad pude ser aprovechada por un atacante para causar algún daño. La vulnerabilidad más habitual es que los datos de un usuario no se validen de forma adecuada. Para esto se pueden usar formularios o webs que se consultan de forma programática.

Inyección de código

En la comunicación entre aplicaciones y bases de datos suelen haber unos tokens como “;” que se pueden utilizar de forma que usando los datos del usuario se puede ejecutar una consulta en una base de datos que no se debería poder ejecutar normalmente. Esta vulnerabilidad era la más importante hasta que en 2021 paso a la tercera posición del OWASP. Existen varios tipos de inyección:

- SQL
- De Log
- Comandos de sistema operativo
- XML
- XPath

Inyección SQL

Se produce cuando los datos de entrada proporcionados por el usuario utilizan algunos de los tokens de SQL de forma que no se procesan de forma adecuada, resultando en que la consulta de validación no se ejecuta de la forma esperada. Por ejemplo, una consulta de login sql puede ser:

```
String q = "SELECT * FROM users Where email='" + email + "'AND password='" + password +"'";
```

Un ejemplo de ataque de inyección sería el siguiente:

```
"SELECT * FROM users Where email = 'bob@acme.com' OR '1' = '1' AND password = 'any'"
```

Al meter un OR de por medio puede engañar al sistema de validación para dejar loguear.

Otro ejemplo sería este:

```
SELECT email, password, full_name
FROM users
WHERE email  = 'any' OR full_name LIKE '%Bob%';
```

También se pueden realizar ataques destructivos metiendo un DROP en la consulta.

Otras técnicas de inyección SQL intentan explotar características de algún gesto de base de datos concreto. Para ello es necesario identificar qué gestor de base de datos almacena la información a base de analizar los mensajes de error.

Otra variante es el Blind SQL Ijection que consiste en que el atacante genera consultas booleanas, de forma que con esto y un diccionario de datos se puede llegar a obtener una base de datos de usuarios completa. Estas vulnerabilidades se pueden probar en vulnweb.

Para prevenir esto, Java tiene consultas parametrizadas, también conocidas como Prepared Statements que evita la construcción de una consulta de forma manual a través de concatenación. Un ejemplo sería el sisguiente en caso de consultar datos usando un correo electrónico:

```
String query 0 "SELECT email, password, full_name FROM users WHERE email =
?";
PreparedStatement stmt = connection.prepareStatement(query);
stmt.setString(1, email);
```

```
string q = "SELECT * FROM users WHERE name = ? AND birthdate > ?";
PreparedStatement preparedStatement =
DriverManager.getConnection(url).prepareStatement(q);
preparedStatement.setString(1, "bob")
preparedStatement.setDate(2, getDate(2000,1,1));
```

En caso de usar Java Persisctence API el formateado de los parámetros de entrada se hace de forma similar.

De todas formas esta solución tiene limitaciones ya que solo se pueden parametrizar valores de columna, pero no identificadores.

por ejemplo en:

```
SELECT * FROM ? WHERE runtime > 90;
```

No se pueden usar consultas parametrizadas.

Si no se pudieran utilizar consultas parametrizadas, otra técnica son listas blancas, que acotan los valores válidos que se pueden utilizar, si los datos recibidos no están en la lista blanca, se rechaza la consulta.

Inyección de JavaScript (XSS)

Entidades externas en documentos XML (XEE)

Deserialización y carga dinámica

From:
https://www.knoppia.net/ - **Knoppia**

Permanent link:
https://www.knoppia.net/doku.php?id=app:vemp&rev=1726682194

Last update: **2024/09/18 17:56**