

Paranoia en la Red

Ataque y defensa

XSS y SQL Injection

Oswaldo Mena Requena
omena@alumnos.inf.utfsm.cl

21 de septiembre de 2009

La gran lección que intenta otorgar el documento presente es que **la seguridad en la Web no es un tema del cual podamos prescindir**... Aún más en el mundo tan interconectado en el que vivimos.

Otro de los objetivos de este documento es otorgar nociones sobre los métodos de ataques más conocidos (XSS y SQL Injection), como protegerse ante ellos y a pensar como un atacante para poder desarrollar software seguro.

Todos los conocimientos entregados, potencialmente podrían utilizarse para atacar tanto a sus compañeros del curso de Bases de Datos (Cosa que ya ha ocurrido entre algunos grupos este semestre), como a portales Web de todo tipo, aún así, *se confía en el criterio del lector para que utilice los conocimientos entregados para construir y no para destruir*.

Este documento ha sido basado en la presentación de Claudio Salazar “Web: Ataque y Defensa” realizada en el ciclo de Charlas Técnicas UTFSM 2007.

1. Introducción

La seguridad en la web es un tópico que no es muy cercano a la gran mayoría de los desarrolladores, sin embargo, con el aumento del uso de Internet, y en particular, con la explosión de la Web 2,0 es necesario preguntarse tanto como usuarios o como desarrolladores: Qué tan seguros nos podemos sentir en la gran Red?

Para responder la pregunta del párrafo anterior, podríamos pensar en las consecuencias que se podrían derivar del inocente acto de no preocuparse de la seguridad:

- Robo de Datos
- Violación de la Privacidad
- Ejecución remota de código malicioso
- Suplantación de contenido (Phishing)
- Cambio de contenido (Deface)

Cerca de un 90 % de los sitios son vulnerables, y un 75 % de los sitios son blancos de XSS. En Chile, la gran mayoría de los atacantes son extranjeros, y como se podrán imaginar, no hay una seria preocupación por ofrecer software seguro y de calidad ¹.

Pero como los sansanos somos tan responsables, éticos, y cuidadosos... Nos preocuparemos de este tema =).

¹Una buena muestra de esto es el deface realizado a caffarena hace un tiempo

2. XSS: Cross Site Scripting

Es un ataque que aprovecha la mala validación de los datos en páginas generadas dinámicamente. Generalmente, el atacante incluye código JavaScript malicioso que posteriormente será ejecutado en el navegador de la víctima.

Existen dos tipos de XSS:

- Persistente: Es código inyectado que queda almacenado y que se ejecutará cada vez que el sitio sea visitado.
- No Persistente: Es código inyectado que aprovecha el paso de parámetros mediante GET. Es decir, el código se inyecta en una URL y luego esa URL se le envía a la víctima. (Algo útil: TinyURL)².

Los sitios que devuelven datos ingresados por el usuario (mediante formularios o similar) son potencialmente vulnerables. Un buen código para probar si un sitio es o no vulnerable a XSS es:

```
<script>alert('XSS');</script>
```

Si aparece por pantalla un MessageBox con el mensaje “XSS”, entonces hemos encontrado un sitio vulnerable a XSS.

Actualmente vivimos en la era de la Web 2.0, esta era se inició gracias a AJAX, lo cual nos ofrece contenidos más dinámicos y atractivos, sin embargo, AJAX extiende el abanico de posibilidades para XSS, lo que nos obliga a tener más cuidado a la hora de diseñar y desarrollar nuestros sistemas.

2.1. Protección

Existen muchas formas de protegerse, probablemente la más simple sea convertir los datos ingresados a HTML entities, es decir, impedimos que el texto se interprete como código HTML, esto se hace con la función `htmlspecialchars` de PHP:

```
$str = 'A 'quote' is <b>bold</b>';  
// Outputs: A 'quote' is &lt;b&gt;bold&lt;/b&gt;  
echo htmlspecialchars($str);
```

3. SQL Injection

En el contexto del curso de Bases de Datos, para nosotros, este tipo de ataque es aún más crítico, ya que el lema de este curso es “La Base de Datos es la pieza fundamental en una organización”.

Un ataque del tipo SQL injection es un ataque que inyecta código SQL en variables ingresadas por el usuario que posteriormente son parte de las consultas que son llevadas al DBMS.

El no verificar nuestras variables, puede llevar a permitir la ejecución de sentencias no deseadas que atenten en contra de la integridad de la BD. Para dar un ejemplo, supondremos el siguiente segmento de código (PHP):

```
$sql = 'select * from usuario where username = '$this->username' and password = '$this->password';';  
$r = pg_query($conexion,$sql);  
if($r) echo 'OK';
```

Es un Login bastante simple, ahora supondremos que esos datos vienen de un formulario (El típico Login), qué pasaría si nos logueamos con los datos `username = “noob”` y `password = “noob’ or 1=1;--”`? La consulta quedaría de la siguiente forma:

```
select * from usuario where username = 'noob' and password = 'noob' or 1=1;--';
```

Con “--” comentamos todo lo que venga después, y como resultado, tendremos que el usuario entrará al sistema de todas formas, sin tener un nombre de usuario y un password válido, aún más, siguiendo la lógica del segmento de código, en el caso de que luego se trate de rescatar la información del usuario, se intentará rescatar la información del primer resultado de la consulta, lo que nos llevará a poder ver los datos de otra persona en el sistema (Como

²Una URL tiene un largo máximo relativamente alto, cercano a los 2000 caracteres, por lo tanto, podríamos incluir código malicioso fácilmente y con soltura

pasa con algunas tareas). Lo anterior es el ejemplo más clásico de SQL Injection.

Ahora veamos un ejemplo más crítico, nos loguearemos con `username="noob"` y con `password="noob";DROP TABLE usuario;--"`, como resultado tendríamos la ejecución de las siguientes consultas:

```
select * from usuario where username = 'noob' and password='noob';
DROP TABLE usuario;--';
```

Se imaginan un sistema que no maneje 10 o 20 usuarios, sino 100000 o más? Es una situación bastante crítica. Y lamentablemente, a la fecha, hay muchos grupos (para no decir todos), que permiten este tipo de ataques.

3.1. Protección

Al igual que en el tipo de ataque anterior, tenemos una amplia gama para obtener seguridad frente a este tema, una posibilidad puede ser la de filtrar los caracteres tales como `;`, `'`, `-` `=`, o bien, mediante expresiones regulares validar los datos de entrada a fin de impedir ataques de este tipo.

Aun así existen funciones que nos permiten un mejor manejo de nuestras consultas, tales como `pg_escape_string()`; (Pre-Formateo de las consultas) o bien `pg_query_params()`; (Consultas parametrizadas).

Se deja al lector la elección de la solución.

4. Políticas de Seguridad

La mejor política es no permitir al usuario la interacción con el sistema, pero como imaginarán no es una solución real a nuestro problema. Sin embargo, la correcta validación de datos, tanto a nivel de entrada como de salida es una excelente política de seguridad para empezar con este tema.