Introducción

En esta lab vamos a aprender a defender nuestras páginas web de ataques de tipo Cross-Site Scripting. Para ello, primero veremos cómo hacer un ataque XSS contra nuestra propia página para después añadir la protección necesaria.

Paso previo: XSS

XSS es una técnica de ataques a servicios web principalmente, pero las páginas web no están totalmente libres de estos ataques.

XSS se basa en la inyección de cualquier tipo de código que permita conseguir el fin malicioso que persigue el atacante, ya sea la obtención de información privilegiada, la difamación de su objetivo, etc…

Como hemos dicho, XSS se basa en la inyección de código, y la única forma que tiene un posible atacante de inyectar código en el flujo de un programa JS es en aquellos puntos donde la página web permita la entrada de datos por parte del usuario. Como programadores web, éstos serán los puntos donde tendremos que prestar atención para estar bien protegidos de este tipo de ataques, y éstos puntos son todos los elementos HTML input y además los parámetros recibidos a través de la URL.

Imaginemos el siguiente caso:

Paso 1: Documento HTML vulnerable

Vamos a crear una página web que muestra un saludo en pantalla al usuario, y el nombre del usuario se recibirá como parámetro en la URL.

```
<!DOCTYPE html>

<html>
<head>
    <meta charset="utf-8" />
    <title>XSS</title>

    <script>
        var name =
decodeURIComponent(window.location.search.substring(1) || "");
        document.write("Hello " + name);
</script>
</head>

<body>

</body>
</html>
```

Éste será el código HTML de nuestra página, como habíamos dicho, cogerá el primer parámetro que encuentre y lo tomará como el nombre del usuario, por ejemplo, si añadimos `?Carlos` al final de la ruta en la barra de direcciones, en la página aparecerá un mensaje de bienvenida que consistirá en Hello Carlos.

Paso 2: XSS

Un comportamiento como éste, puesto en ojos de un hacker le hará pensar que nuestro documento web genera contenido dinámicamente a partir del parámetro que recibe en la URL, y, difícilmente podrá reprimir la tentación de intentar un ataque XSS contra nuestro sitio.

Lo primero que probará es, si efectivamente, el sitio web, es vulnerable a inyección de contenido a través de parámetros en la URL.

Antes de analizar en profundidad la página web para determinar qué información le puede interesar, o cómo quiere conseguir su objetivo malicioso, comprobará si realmente existe una brecha de seguridad en el punto que ha descubierto, para ello, intentará la inyección de algo simple, algo como esto:

```
?%3Cscript%3E alert(%27vulnerable%27) %3C/script%3E
```

Con este contenido pasado como parámetro, la página muestra un Alert Box confirmando la sospecha de que el documento web es vulnerable a inyección de código.

Lo que está ocurriendo es que el contenido generado es la creación de un nuevo script JS con código y, como ya sabemos, al ser invocado directamente desde el propio documento, tiene total libertad para manipular el contenido que desee o manejar, por ejemplo, eventos de pulsación de teclas para conseguir passwords, o robar cookies almacenadas en el navegador para poder hacer peticiones a un servidor en nombre del usuario sin necesidad, si quiera, de conseguir su password.

Este tipo de ataques son muy frecuentes en enlaces de una página a otra. Si alguien enlaza vuestra página desde la suya y en el atributo href del enlace hacia vuestra página coloca un script de este tipo, cuando el usuario vaya a visitar vuestra página desde la que os enlaza estará siendo atacado.

Paso 3: Protegiendo la vulnerabilidad

La forma de prevenir estos ataques es filtrar los tags HTML que puedan existir en cualquier contenido que provenga de un tercero y que usemos para generar contenido dinámico en nuestra web, por ejemplo, modificando nuestra página así:

```
document.write("Hello " + escape(name));
```

Con este código simplemente escapamos todo el contenido recibido de la url para prevenir la inyección de código malicioso. Esta situación es tan común que se ha incluido la función que se encargue de escapar los tags HTML en el propio Core de JS.

Podéis probar que la funcionalidad sigue funcionando correctamente cuando el parámetro es un nombre de un usuario pero “se activa” cuando el código es potencialmente peligroso.