Moises Rodriguez Aguilar 1837462
Miguel Salazar S.
4 de septiembre de 2017

# Vulnerabilidades
## HTML y JavaScript

Los ataques XSS se basan en la inyección de scripts maliciosos que se ejecutan en el navegador del usuario. En muchos casos, el usuario ni siquiera es consciente de lo que está ocurriendo. Los tipos principales de ataques XSS son:

•*Reflected*: Normalmente se basan en conseguir que el usuario siga un link o complete un formulario malicioso aunque aparentemente normal. La respuesta del servidor web contendrá el script inyectado que se ejecutará en el navegador del cliente. Es el tipo de ataque en el que se basa el ejemplo que veremos.

•*Stored*: La diferencia con el anterior es que el código malicioso se inyecta en algún elemento persistente, como una base de datos, de manera que se consigue que se ejecute en el navegador del cliente al recuperar la información almacenada.

## Detectando vulnerabilidades XSS con ZAP de OWASP

La herramienta ZAP de OWASP está basada en el proyecto Paros Proxy. Por lo tanto, no es de extrañar que, entre sus muchas funcionalidades, se encuentre el poder usarse como proxy, almacenando los datos de navegación, y permitiendo realizar diversas operaciones sobre los mismos. En nuestro caso, usaremos el escaneo activo que es capaz de detectar varios tipos de vulnerabilidades, entre ellas las del tipo XSS.

Los pasos a seguir para realizar dicho escaneo sobre nuestra página JSP (xss_html.jsp) son muy sencillos. En primer lugar, se debe arrancar la herramienta ZAP e indicarle el puerto donde escuchará el proxy que incluye. A continuación, modificaremos la configuración de nuestro navegador web para que use como proxy ZAP, indicando el puerto correspondiente.

## Explotando la vulnerabilidad

Una vez ZAP nos ha indicado que amount es vulnerable, vamos a intentar inyectar código javascript que nos desplegará una ventana emergente con los datos de la tarjeta. Para ello, modificaremos el evento onsubmit del formulario de manera que recopile los datos de la tarjeta introducida en dicho formulario, nos los presente (aunque un atacante podría haber intentado enviarlos a algún sitio) y, finalmente, que llame a la función que se llamaba anteriormente para que el usuario no note nada extraño.

Además, con el fin de evitar que el usuario note que ocurre algo extraño, cerramos las " del atributo value donde se inserta el valor de amount, y que es donde inyectaremos nuestro código, y también cerramos el campo input añadiendo la secuencia />. A continuación, añadimos el código javascript que muestre la info de la tarjeta. Finalmente, añadimos la cadena <br que quedará cerrada con el /> que viene a continuación (HTML que cierra el input).

## Solucionar la vulnerabilidad

OWASP también proporciona la librería ESAPI, disponible para varios lenguajes, que permite, entre otras muchas cosas, validar los caracteres recibidos como entrada del usuario o codificar adecuadamente los caracteres según dónde se vayan a utilizar: HTML, atributo HTML, javascript, …

En nuestro caso, únicamente nos preocuparemos únicamente de codificar correctamente los datos de entrada para que no sean interpretados como código javascript por el navegador.

Para ello, creamos una copia del JSP anterior, al que llamamos **xss_html_esapi.jsp**, y en el que reemplazamos la línea:

```
final String amount = request.getParameter("amount");
```

Por:

```
final String amount =
ESAPI.encoder().encodeForHTML(request.getParameter("amount"));
```

Si ahora probamos de nuevo a inyectar nuestro código, obtenemos un resultado distinto: de entrada, aparecen caracteres extraños en el input donde introducir el **amount**, lo que ya da una pista de que algo ha ido mal.

## JavaScript

En teoría, el ataque por inyección es sencillamente una forma de "colar" sentencias que no suelen pasar de una broma, pero que convenientemente situadas con un poco de malicia, podrían obtener nuestros datos.

Por poner un ejemplo, la web del **Banco Santander** (Este fallo ya fue notificado y está actualmente corregido).

Una persona cualquiera va a ver su saldo en el Banco Santander y a realizar una transferencia. La web del Banco Santander tendría una URL como ésta:

www.bancosantander.com/mirarCuenta?opeowpoe=123

Como al equipo de personas que construyó inicialmente la web no tomó en cuenta validar los datos de entrada (las validaciones constituyen una parte vital para la creación de una web en cuestiones de seguridad y desde el punto de vista de un desarrollador) un atacante podría construir algo así: (la forma del ataque está cambiada para evitar a los Script Kiddies)

www.bancosantander.com/mirarCuenta?opeowpoe=123(javascript:alert("es usted un ladrón");)

Y la víctima, a la que previamente se le habría enviado dicho enlace, vería lo siguiente:

Un enlace **válido** que llevaría de inicio un dominio conocido y en el que confía: **www.bancosantander.com**y en este caso inocente, un mensaje ridículo que no tienen mayores consecuencias.

Ahora imaginad lo siguiente: el mismo escenario pero esta vez el pirata informático va mas lejos: crea un código por Javascript que muestra una pantalla de login de usuario y contraseña **idéntica** a la real del Banco Santander y que por debajo envía dichos datos a un correo que previamente será consultado por el pirata y que luego redirige a la verdadera página de login del Banco Santander sin que el usuario perciba nada extraño. Este escenario fue el que se probó (de forma interna) y dio un resultado excelente: muchos usuarios dieron sus datos de inicio de sesión.

Otro escenario real: una conocida tienda online (no diré el nombre porque su error no está corregido) transmitía el precio de sus artículos a su sistema de pedidos automático por la URL de esta manera:

www.todobaratoybonito.com?nombre=televisorPlasma&precio=1000&

Al ver ese tipo de URL, la prueba era sencilla: pusimos en la variable precio el valor que quisimos, es decir:

www.todobaratoybonito.com?nombre=televisorPlasma&precio=1&

Y el resultado fue que en la cesta de la compra del usuario, a falta de darle al botón enviar, existía un televisor de plasma de 1000 euros que pudimos comprar por 1 €.

Como anécdota, un pirata informático que intentó aprovecharse de dicha vulnerabilidad fue arrestado por mandar el paquete al lado de su casa.

# CONCLUSION

En conclusión muchos de los errores y fallos se pueden evitar validando la información ya que en muchas ocaciones el usuario no se da cuenta, los programadores web que cometen este tipo de errores no quiere decir que sean malos haciendo su trabajo si no que la presión por tener listo el servicio lo antes posible hace que se cometan errores que los piratas informáticos encuentra y aprovechan, por otra parte en otras ocaciones es culpa del usuario que no tiene en conciencia ningún sentido de seguridad ya que entra a cualquier link, no comprueba la versada de estos o información que recibe por correo.