UNAM CERT

Cross-Site Scripting (XSS)

Por Angie Aguilar Domínguez

XSS es un ataque de inyección de código malicioso para su posterior ejecución que puede realizarse a sitios web, aplicaciones locales e incluso al propio navegador.

Sucede cuando un usuario mal intencionado envía código malicioso a la aplicación web y se coloca en forma de un hipervínculo para conducir al usuario a otro sitio web, mensajería instantánea o un correo electrónico. Así mismo, puede provocar una negación de servicio (DDos).

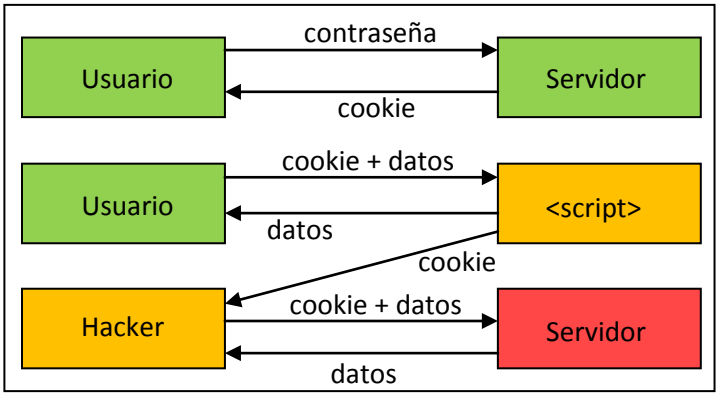

Fig. 1. Operación de un ataque XSS

Generalmente, si el código malicioso se encuentra en forma de hipervínculo es codificado en HEX (basado en el sistema de numeración hexadecimal, base 16) o algún otro, así cuando el usuario lo vea, no le parecerá sospechoso. De esta manera, los datos ingresados por el usuario son enviados a otro sitio, cuya pantalla es muy similar al sitio web original.

De esta manera, es posible secuestrar una sesión, robar cookies y cambiar la configuración de una cuenta de usuario.

Tipos de Ataques.

Las diversas variantes de esta vulnerabilidad pueden dividirse en dos grandes grupos: el primero se conoce como XSS persistente o directo y el segundo como XSS reflejado o indirecto.

Directo o persistente. Consiste en invadir código HTML mediante la inclusión de etiquetas <script> y <frame> en sitios que lo permiten.

Local. Es una de las variantes del XSS directo, uno de sus objetivos consiste en explotar las vulnerabilidades del mismo código fuente o página web. Esas vulnerabilidades son resultado del uso indebido del DOM (Modelo de Objetos del Documento, es un conjunto estandarizado de objetos para representar páginas web) con JavaScript, lo cual permite abrir otra página web con

código malicioso JavaScript incrustado, afectando el código de la primera página en el sistema local. Cuando el XSS es local, ningún código malicioso es enviado al servidor. El funcionamiento toma lugar completamente en la máquina del cliente, pero modifica la página proporcionada por el sitio web antes de que sea interpretada por el navegador para que se comporte como si se realizara la carga maliciosa en el cliente desde el servidor. Esto significa que la protección del lado del servidor que filtra el código malicioso no funciona en este tipo de vulnerabilidad.

Indirecto o reflejado. Funciona modificando valores que la aplicación web pasa de una página a otra, sin emplear sesiones. Sucede cuando se envía un mensaje o ruta en una URL, una cookie o en la cabecera HTTP (pudiendo extenderse al DOM del navegador).

Consideraciones.

Como desarrollador.

La aplicación web que se desee implementar debe contar con un buen diseño. Posteriormente, se deben realizar diversos tipos de pruebas antes de su liberación, para detectar posibles fallos y huecos de seguridad, mediante el empleo de alguna herramienta automatizada. También, es conveniente proporcionar mantenimiento a la aplicación y estar actualizado en las versiones de las herramientas que se emplean para su puesta en marcha.

Algunas recomendaciones para mitigar el problema, son:

Emplear librerías verificadas o algún framework que ayude a disminuir el inconveniente. Por ejemplo: la librería anti-XSS de Microsoft, el módulo ESAPI de codificación de OWASP, Apache Wicket, entre otros.

Entender el contexto en el cual los datos serán usados y la codificación de los mismos, este aspecto es importante cuando se envían datos de un componente a otro de la aplicación o cuando se deben enviar a otra aplicación.

Conocer todas las áreas potenciales donde las entradas no verificadas pueden acceder al software: parámetros o argumentos, cookies, información de la red, variables de entorno, resultados de consultas, búsqueda de DNS reversible, peticiones enviadas en las cabeceras, componentes de la URL, correos electrónicos, archivos, nombres de archivo, bases de datos o algún sistema externo que proporcione información a la aplicación.

Las validaciones de datos de entrada, deben realizarse siempre del lado del servidor, no sólo en el lado del cliente. Los atacantes pueden evitar la validación realizada del lado del cliente modificando valores antes de realizar verificaciones o remover por completo esta validación.

En caso de ser posible, emplear mecanismos automatizados para separar cuidadosamente los datos del código fuente: revisión de comillas, codificación y validación automática que muchas veces se escapan al desarrollador.

Por cada página web generada, se recomienda emplear una codificación determinada de caracteres, ya que si no se especifica, el navegador puede dar un trato diferente a ciertas secuencias de caracteres especiales, permitiendo la apertura del cliente a posibles ataques.

Para mitigar el problema de ataque contra el uso de cookies, es conveniente indicar que tiene el formato de HttpOnly. En los navegadores que lo soportan, puede prevenirse que la cookie sea usada por scripts maliciosos desde el lado del cliente.

Se debe emplear una estrategia de validación de las entradas: rechazar aquellas que no cumplan con lo especificado, limpiar las que sean necesarias. Al validar, considérense las características de cada entrada: longitud, tipo de dato, rango de valores aceptados, entradas perdidas o adicionales, sintaxis, consistencia con otras entradas proporcionadas y seguimiento de las reglas del negocio.

Cuando se construyan páginas web de forma dinámica (generadas de acuerdo a las entradas o solicitudes de los usuarios), es recomendable usar listas blancas estrictas. Todas las entradas deben ser limpiadas y validadas, incluidos cookies, campos ocultos, cabeceras y la propia dirección.

Cuando una cantidad aceptable de objetos, como nombres de archivo o URL es limitada o conocida, es conveniente crear una conjunto de asignaciones de valores de entrada fijo a los nombres de archivo o URL y rechazar todos los demás.

Se recomienda usar un firewall de aplicaciones capaz de detectar ataques cuando el código se genere dinámicamente, como medida de prevención, debe complementarse con otras para proporcionar defensa en profundidad.

Como Administrador de Bases de Datos.

Así como el desarrollador debe validar las entradas proporcionadas por parte del usuario, el encargado de diseñar e implementar la base de datos debe considerar la seguridad de la misma, pues en ella se guarda la información proporcionada por los usuarios y es manipulada mediante la aplicación web. Los datos que serán almacenados, también pueden ser validados mediante el uso de *constraints* (restricciones aplicables a los objetos de una base de datos: unique, default, not null, check) que restringen la entrada para cada campo.

PDF XSS

Es una vulnerabilidad ampliamente usada para afectar el Acrobat Reader de Adobe. En este caso, si se abusa de las características para abrir archivos en Acrobat, un sitio bien protegido se vuelve vulnerable a un ataque de tipo XSS si da alojamiento a documentos en formato PDF.

Esto afecta seriamente, a menos que se actualice el Reader o se cambie la forma en que el navegador maneja dichos documentos.

Una manera de combatirlo, si se cuenta con el servidor de aplicaciones web Apache, es llevar a cabo la correcta configuración de ModSecurity, ya que cuenta con directivas de protección para archivos en formato PDF.

XSRF

Un ataque Cross-Site Request Forgery (XSRF ó también CSRF) explota la confianza que un usuario tiene hacia las entradas proporcionadas por un sitio.

Por ejemplo: un usuario se encuentra autenticado y navegando en un sitio, en ese momento un atacante obtiene el control de su navegador, con él realiza una solicitud a una tarea de una URL válida del sitio, por lo que el atacante tendrá acceso como si fuera el usuario previamente registrado.

Distintivamente, un atacante intercalará código HTML o JavaScript malicioso en un correo o en una tarea específica accesible desde una URL, que se ejecuta ya sea directamente o empleando un error de tipo XSS. También, es posible realizar inyección a través de lenguajes como el BBCode. Este tipo de ataques son difíciles de detectar.

Muchas de las funcionalidades de un sitio web son susceptibles de uso durante un ataque XSRF. Esto incluye información enviada tanto por GET como por POST.

También puede usarse como vector para explotar vulnerabilidades de tipo XSS en una aplicación. Ejemplos de ello son: una vulnerabilidad de tipo XSS en un foro donde un atacante puede obligar al usuario a publicar –sin que éste se dé cuenta- un gusano informático. Un atacante puede también usar XSRF para transmitir un ataque a algún sitio de su elección, así como realizar un DDos.

Sin embargo, las formas más comunes de realizar este tipo de ataque consisten en usar la etiqueta HTML o el objeto JavaScript empleados para imágenes. Distintivamente, el atacante infiltrará un email o sitio web en ellos, así cuando el usuario cargue la página o el correo electrónico, también estará realizando la petición a la URL que haya colocado el atacante.

Un atacante puede instalar su script dentro de un documento de Word, un archivo de flash, un clip de video, redifusión web RSS o Atom, o algún otro tipo de formato que pueda alojar el script.

Si un sitio web permite ejecutar sus funciones empleando una URL estática o peticiones POST, es posible que sea vulnerable, si la función se lleva a cabo mediante la petición GET, el riesgo es mayor. Si se realizan las mismas funciones, de la misma forma repetidamente, entonces la aplicación puede ser vulnerable.

Un ataque XSRF no puede evitarse mediante la verificación del *referer* de las cabeceras de la petición realizada, ya que puede “limpiarse” o modificarse mediante algún tipo de filtro. Las cabeceras pueden falsearse usando XMLHTTP, por ejemplo.

Una de las soluciones más conocidas, consiste en adjuntar un *token* no predecible y cambiante a cada petición. Es importante que el estado de éste vaya asociado con la sesión del usuario, de otra manera un atacante puede adjuntar su propio *token* válido y emplearlo en su beneficio. Adicionalmente, al ligarlo a la sesión del usuario es importante limitar el periodo durante el cual será válido.

Referencias.

Mesografía.

http://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
http://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OWASP-DV-003)
http://www.modsecurity.org/projects/modsecurity/apache/feature_universal_pdf_xss.html
http://www.cgisecurity.com/csrf-faq.html
http://www.isecpartners.com/files/XSRF_Paper_0.pdf
http://www.techrepublic.com/blog/security/what-is-cross-site-scripting/426
http://cwe.mitre.org/data/definitions/79.html