UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

LICENCIATURA EN SEGURIDAD EN TECNOLOGÍAS DE LA INFORMACIÓN

DISEÑO ORIENTADO A OBJETOS

RIESGOS Y/O VULNERABILIDADES EN APLICACIONES WEB (HTML y JAVASCRIPT)

CATEDRÁTICO: MIGUEL ÁNGEL SALAZAR

ALUMNO: SALVADOR GONZÁLEZ GARCÍA

GRUPO: 006

MATRÍCULA: 1690539

Introducción.

En este ensayo se hablará de las principales vulnerabilidades que tienen las aplicaciones web debido al uso de HTML y Javascript; ya que tanto el uso de estos lenguajes es muy útil a la hora del desarrollo de un sitio web, tanto como lo puede ser muy peligroso. A continuación, veremos algunos tipos de vulnerabilidades que existen.

RIESGOS Y/O VULNERABILIDADES EN APLICACIONES WEB (HTML y JAVASCRIPT)

Los sitios web o aplicaciones web que visitamos pueden ser estáticos o dinámicos.

En cuanto a los sitios estáticos, se distinguen por ser simples y económicos, esto es porque son usados para hacer una descripción acerca de "quienes somos". Estas, no permiten la presencia de aplicaciones en las mismas.

Ahora bien, los sitios dinámicos, por el contrario, son más complejos, ya que permite la interacción de un usuario con un sitio web.

De este modo, Javascript es un lenguaje de programación que fue desarrollado para hacer que las páginas estáticas se comporten o tengan un comportamiento como las estáticas, ya que permite ejecutar códigos automáticamente debido a la interacción que tiene con el navegador elegido para visitar la página web. Es por esto que una página puede ser modificada, no obstante, de estar programada en HTML.

Por lo que, la forma de ejecutar códigos puede ser más atractiva para el sitio y para el usuario por cuestiones de velocidad, costos y la posibilidad de saltar pasos molestos, sucede que, para un atacante, JavaScript es ideal para realizar funciones benignas como lo es el: saltar herramientas de seguridad.

Por ejemplo:

```
// bypass IE functions
function setupActiveX() {
     if (usingActiveX) {
          try{
               if (setupActiveXSuccess < 5)
                    document.writeln('<DIV><
STYLE="position: absolute; top: 0; left:
                    document.write('<INPUT S
                    popWindow=window.createP
                    popWindow.document.body.
HEIGHT=1 DATA="'+myurl+'/popuppopup.html
                    setupActiveXSuccess = 6;
               }
          }
          catch(e){
               if (setupActiveXSuccess < 5)
                    setupActiveXSuccess++;se
               }else if (setupActiveXSucces
                    activeXTried = true;setu
               }
          }
     }
}
function tryActiveX(){if (!activeXTried
document.getElementById('OurPopupObject'
window.open('about:blank','Ads','scrollb
OurPopupObj.DOM.Script.paypopupPop();if
popWindow && popWindow.document.getEleme
popWindow.document.getElementById('getPa
{myWindow=popWindow.document.getElementB
```

Este código es utilizado para ejecutar ActiveX sin la intervención del usuario. Por supuesto, esta acción sólo funciona en el navegador Internet Explorer.

En cambio, este otro código es utilizado para evitar que ciertos programas tradicionales de seguridad como detectores de pop-pups, alerten al usuario acerca de las acciones que lleva a cargo el sitio web dañino:

```
function blockError(){return true;}
window.onerror = blockError;
//bypass [illegible] internet security popup blocker
if (window.SymRealWinOpen){window.open = SymReal
if (window.NS_ActualOpen) {window.open = NS_Actu
if (typeof(usingClick) == 'undefined') {var usir
if (typeof(usingActiveX) == 'undefined') {var us
if (typeof(popwin) == 'undefined') {var popwin =
if (typeof(poped) == 'undefined') {var poped = f
var blk = 1;
var setupClickSuccess = false;
var googleInUse = false;
var pop = 'enter';
var myurl = document.location.protocol + "//" +
```

Este JavaScript, al ejecutarse, descarga automáticamente un troyano y el bloqueador de pop-pup no se entera de lo sucedido.

Otra ventaja que encuentran los cibercriminales en la utilización de JavaScript es que sus códigos pueden explotar múltiples vectores al mismo tiempo. En otras palabras, no hace falta propagar el código a varios sitios; con tan solo estar presente en un sitio de uso masivo, es posible atacar a muchas víctimas y robar nombres de usuarios y contraseñas o capturar contraseñas ingresadas en el teclado, entre otras cosas.

Las vulnerabilidades más comunes en una aplicación web son:

Cross-Site Scripting (XSS)

Esta es una vulnerabilidad permite que, utilizando los parámetros de entrada de la aplicación, modificar y añadir código a la misma.

Son vulnerabilidades que se encuentran en el servidor, pero que están destinadas a atacar al cliente.

Generalmente, se necesita que el cliente sigua un enlace de la aplicación, en el que se ha modificado algún parámetro, para permitir añadir código, que se ejecutará en el navegador del cliente.

Normalmente, el código inyectado será html o JavaScript y la intención será un robo de cookies del cliente, predicción del id de sesión, etc.

Esta vulnerabilidad se aprovecha de la confianza en el cliente en el sitio web: Verá que es el dominio de la aplicación y que, al seguir el enlace, llega realmente al sitio web que quería, no hay suplantación del mismo.

La forma de corregir esta vulnerabilidad es filtrar y validar todas las entradas de la aplicación. No se debe utilizar nunca una variable que se recibe desde el cliente, confiando en que ésta tendrá un valor correcto. Los lenguajes de programación incluyen diferentes funciones que permiten filtrar el contenido de las variables, dependiendo de dónde las vayamos a utilizar.

Cross Site Request/Reference Forgery (CSRF)

Esta vulnerabilidad es una evolución de los XSS. En este caso, se va a explotar la confianza en el servidor sobre el cliente. Es decir, nos haremos pasar por un cliente legítimo, utilizando datos parciales del mismo.

Esta vulnerabilidad está presente en formularios. Cuando estos se envían al servidor, es necesario asegurarse que la petición es legítima y debemos asegurarnos que el cliente ha realizado la petición, realizando los pasos previos necesarios.

La forma más común para eliminar esta vulnerabiliad o, al menos, mitigarla, es la inclusión de tockens dinámicos. En los actuales FrameWorks, suelen incluirse mecanismos para añadir esta protección a los formularios, de una forma muy sencilla. En algunos, por ejemplo, Laravel (utilizado para desarrollo de aplicaciones en PHP), basta añadir una etiqueta a la plantilla del formulario, para que se añada al mismo el tocken anti-csrf.

SQL INJECION

Esta, una vulnerabilidad que permite a un atacante realizar consultas a una base de datos, se vale de un incorrecto filtrado de la información que se pasa a través de los campos y/o variables que usa un sitio web, es por lo general usada para extraer credenciales y realizar accesos ilegítimos. Un fallo de este tipo puede llegar a permitir ejecución de comandos en el servidor, subida y lectura de archivos, o peor aún, la alteración total de los datos almacenados.

Para prevenir los SQLi, se deben parametrizar las consultas y es necesario filtrar y comprobar el valor de las entradas. También es muy importante restringir al máximo los permisos del usuario con el que la aplicación se conecta a la base de datos y, por supuesto, para cada base, utilizar un usuario diferente. Esto es muy importante y, aunque es una mala práctica usar una misma instancia del motor de base de datos para diferentes aplicaciones, es un escenario muy común. Por este motivo, si un usuario de la base de datos tiene permisos sobre varias de diferentes aplicaciones y una de éstas presenta esta vulnerabilidad, el atacante podría obtener los datos del resto de aplicaciones.

Además de esto, es muy importante ocultar los errores provocados por consultas en base de datos. Esto es porque la forma de detectar un SQLi es intentar forzar un error (la típica ‘) y, una vez se comprueba que existe, si se muestra al usuario, éste puede extraer información, como el motor de base de datos usado, etc., con lo que se le facilitará la realización del ataque.

Referencias

https://www.welivesecurity.com/la-es/2014/09/25/ataque-jquery-javascript-arma-doble-filo/

https://www.welivesecurity.com/la-es/2008/03/04/saltando-herramientas-seguridad-javascript/

https://hacking-etico.com/2017/04/04/las-principales-vulnerabilidades-web/