Práctica 2

Vulnerabilidades de Seguridad - Primavera 2017

Presentación

La práctica 2 de vulnerabilidades de seguridad trata sobre vulnerabilidades en aplicaciones web. Una vez resueltos los ejercicios, recordad hacer las siguientes comprobaciones:

	Lo he hecho?
La solución se tiene que entregar al registro de evaluación continuada en un único archivo en formato pdf. No se admite ninguno otro formato (doc, docx, odt, zip, raro, tar.gz, ...).	☐
Todas las páginas tienen que estar numeradas y tienen que contener el nombre y los apellidos.	☐
La fecha máxima de entrega es el **12 de Mayo de 2016 (a las 24 horas UTC/GMT + 1 hora).**	☐
Esta actividad se puede resolver, con la nota máxima, de forma relativamente breve. En ningún caso la extensión de la solución tiene que superar las **10 páginas** con fuente 10pt (incluido el texto en las figuras) con interlineado simple. Por favor, limitad el uso de las capturas de pantalla para cuando sean estrictamente necesarias y mirad que no ocupen demasiado espacio a las páginas.	☐
Razonad la respuesta a todos los ejercicios e indicad todos los pasos que habéis realizado para obtener la solución. Las respuestas sin justificación, que sean una copia de una fuente de información y/o que no contengan las referencias utilizadas, no recibirán ninguna puntuación.	☐

Contexto

Nos encontramos con una aplicación de “scouting” de deportistas de la empresa ACME, en esta aplicación, los diferentes usuarios pueden añadir deportistas que los resulten interesantes a la base de datos, y anexar a cada uno de estos los comentarios que sean oportunos sobre la idoneidad de su contratación. Como curiosidad, los comentarios no pueden ser editados, pero los deportistas sí (por si cambian de equipo).

Esta aplicación web presenta varias vulnerabilidades que se estudian a continuación.

Configuración

La web sobre la que trata este ejercicio se puede encontrar al archivo "web.tar.gz" que os podéis descargar del enlace http://deic.uab.cat/~asanchezc/web.tar.gz . Para hacerla funcionar se puede usar un servidor web apache con php y sqlite3. En Debian/Ubuntu los paquetes a instalar son: sqlite3, php5-sqlite, php5, y apache2. Para Windows o OSX, se recomienda usar XAMPP, en este caso, hay que activar la extensión SQLLite3 modificando el archivo xampp/php/php.ini. En general, un error habitual a la hora de ponerla en funcionamiento es no dar al archivo de base de datos los permisos necesarios.

Los archivos dentro de web.tar.gz están separados entre los que son accesibles a través del servidor web y los que no lo son (todos son accesibles para el servidor web, pero no todos los sirve en Internet). Se entiende que los archivos de la carpeta “web” están situados en un host cualquiera (por ejemplo,

http://localhost:80). Los archivos que hay en la raíz son accesibles y los que están en la carpeta “private” no lo son, por lo tanto, **ninguna solución puede basarse en información que se haya obtenido a partir de los archivos de la carpeta “private” excepto que el enunciado lo pida explícitamente**.

Podéis comprobar que efectivamente el fichero que os descargáis corresponde con el enunciado de esta práctica validando el hash SHA256:
37f0ac0650e7025741a24aedd2f2cdf9f502d3ab57f6dc0b2760c1bca4dcc09d.

Pregunta 1: SQL Injection

La página no permite añadir jugadores a usuarios no autenticados, un formulario nos exige que introduzcamos un usuario y contraseña válidos. Lo primero que haremos es comprobar que este formulario es vulnerable a una inyección y aprovecharlo para saltarnos esta protección.

a) Dad un ejemplo de combinación de usuario y contraseña que provoque un error en la consulta SQL generada por este formulario. A partir del mensaje de error obtenido, decid cuál es la consulta SQL que se ejecuta, cual de los campos introducidos al formulario utiliza y cual no.

Escribo el valores...	
En el campo...	
Del formulario de la página...	
La consulta SQL que se ejecuta es...	
Campos del formulario web utilizados en la consulta SQL...	
Campos del formulario web no utilizados en la consulta SQL...	

b) Gracias a la SQL Injection del apartado anterior, sabemos que este formulario es vulnerable y conocemos el nombre de los campos de la tabla “users”. Para tratar de impersonar a un usuario, nos hemos descargado un diccionario que contiene algunas de las contraseñas más utilizadas (se listan a continuación):

password

123456

12345678

1234

qwerty

12345678

dragon

Dad un ataque que, utilizando este diccionario, nos permita impersonar un usuario de esta aplicación y acceder en nombre suyo. Tened en cuenta que no sabéis ni cuántos usuarios hay registrados en la aplicación, ni los nombres de estos.

Explicación del ataque:	El ataque consiste en repetir...
	…utilizando en cada iteración una contraseña diferente del diccionario.
Campo de usuario con que el ataque ha tenido éxito:	
Campo de contraseña con que el ataque ha tenido éxito:	

c) Si vais **a private/auth.php**, veréis que en la función “areUserAndPasswordValid”, se utiliza “SQLite3::escapeString()”, pero, aún así, el formulario es vulnerable a SQL Injections, explicad cuál es el error de programación de esta función y como lo podéis corregir.

Explicación del error...	
Solución: cambiar la línea con el código...	
… por la línea siguiente...	

d) Si habéis tenido éxito con la apartado b), os habéis autenticado utilizando el usuario “luis” (si no habéis tenido éxito, podéis utilizar la contraseña “1234” para realizar este apartado). Con el objetivo de mejorar la imagen del jugador “Iouga”, le queremos escribir un buen puñado de comentarios positivos, pero no los queremos hacer todos con la misma cuenta de usuario.

Para hacer esto, en primer lugar habéis hecho un ataque de fuerza bruta sobre el directorio del servidor web (por ejemplo, probando nombres de archivo) y habéis encontrado el archivo “add_comment.php~”. Estos archivos seguramente se han creado como copia de seguridad al modificar el archivo “.php” original directamente al servidor. En general, los servidores web no interpretan (ejecuten) los archivos “.php~” sino que los muestran como archivos de texto sin interpretar.

Esto os permite estudiar el código fuente de “add_comment.php” y encontrar una vulnerabilidad para publicar mensajes en nombre otros usuarios. Cuál es esta vulnerabilidad, y cómo es el ataque que utilizáis para explotarla?

Vulnerabilidad detectada...	
Descripción del ataque...	

Pregunta 2: XSS

En vistas de los problemas de seguridad que habéis encontrado, empezáis a sospechar que esta aplicación quizás es vulnerable a XSS (Cross Site Scripting).

a) Para ver si hay un problema de XSS, crearemos un comentario que muestre un alert de Javascript siempre que alguien consulte els comentarios de aquel jugador (show_comments.php).

Dad un mensaje que genere un «alert» de Javascript al consultar el listado de mensajes.

Introduzco el mensaje...	
En el formulario de la página...	

b) Por qué dice "&" cuando miráis un link (como el que aparece a la portada de esta aplicación pidiendo que realices un donativo) con parámetros GET dentro de código html si en realidad el link es sólo con "&" ?

Explicación...	

c) Explicad cuál es el problema de show_comments.php, y como lo arreglaríais. Para resolver este apartado, podéis mirar el código fuente de esta página.

Cuál es el problema?	
Sustituyo el código de la/las líneas...	
Por el siguiente código...	

d) Descubrid si hay alguna otra página que esté afectada por esta misma vulnerabilidad. En caso positivo, explicad como lo habéis descubierto.

Otras páginas afectadas...	
Cómo lo he descubierto...	

Pregunta 3: XSRF

Ahora ya sabemos que podemos realizar un ataque XSS. Hemos preparado el siguiente enlace: *http://web.pagos/donate.php?amount=100&receiver=attacker*, mediante el cual, cualquiera que haga click hará una donación de 100€ al nuestro usuario (con nombre 'attacker') de la famosa plataforma de pagos online 'web.pagos' (Nota: como en realidad esta es una dirección inventada, vuestro navegador os devolverá un error 404).

a) Editad un jugador para conseguir que, en el listado de jugadores (list_players.php) aparezca, debajo del nombre de su equipo y antes de "(show/add comments)" un botón llamado "Profile" que corresponda a un formulario que envíe a cualquiera que haga click sobre este botón a esta dirección que hemos preparado.

En el campo...	
Introduzco...	

b) Una vez lo tenéis terminado, pensáis que la eficacia de este ataque aumentaría si no necesitara que el usuario pulse un botón. Con este objetivo, cread un comentario que sirva vuestros propósitos sin levantar ninguna sospecha entre los usuarios que consulten los comentarios sobre un jugador (show_comments.php).

Introduzco el mensaje...	
En el formulario de la página...	

c) Pero 'web.pagos' sólo gestiona pagos y donaciones entre usuarios registrados, puesto que, evidentemente, le tiene que restar los 100€ a la cuenta de algún usuario para poder añadirlos a nuestra cuenta.

Explicad qué condición se tendrá que cumplir por que se efectúen las donaciones de los usuarios que visualicen el mensaje del apartado anterior o hagan click en el botón del apartado a).

Condición a cumplir...	

d) Si 'web.pagos' modifica la página 'donate.php' para que reciba los parámetros a través de POST, quedaría blindada contra este tipo de ataques? En caso negativo, preparad un mensaje que realice un ataque equivalente al de la apartado b) enviando los parámetros "amount" i "receiver" por POST.

Queda blindada?	
Mensaje que realiza el ataque (si se tercia)...	

que sea no impide específicamente suyo uso en el marco de la práctica. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por el copyright.

Nota: esto se refiere al copyright del documento que entregáis al registro de evaluación continua y no al copyright de las herramientas que podáis haber usado. Por ejemplo, al usar una imagen en la respuesta de un ejercicio tenéis que seguir el que aquí se indica, pero por el contrario, si usáis el sistema operativo Debian para resolver el enunciado, entonces no es necesario.