

 hdiv

1.0

Referencia



1.	<u>INTRODUCCION</u>	4
1.1	PARAMETER TAMPERING	4
1.2	SQL-INJECTION	6
1.3	CROSS-SITE SCRIPTING (XSS)	7
2.	<u>ESTADO DEL ARTE</u>	10
3.	<u>LIBRERIA DE TAGS</u>	11
3.1	INTRODUCCIÓN	11
3.2	CONCEPTOS BASE	11
3.3	ARQUITECTURA	13
4.	<u>ESTRATEGIAS DE FUNCIONAMIENTO</u>	14
4.1	ESTRATEGIA DE CIFRADO	15
4.1.1	INTRODUCCIÓN	15
4.1.2	GENERACIÓN DE LA RESPUESTA	15
4.1.3	VALIDACIÓN	16
4.2	ESTRATEGIA HASH	17
4.2.1	INTRODUCCIÓN	17
4.2.2	GENERACIÓN DE LA RESPUESTA	18
4.2.3	VALIDACIÓN	18
4.3	ESTRATEGIA DE MEMORIA	19
4.3.1	INTRODUCCIÓN	19
4.3.2	GENERACIÓN DE LA RESPUESTA	19
4.3.3	VALIDACIÓN	19
5.	<u>LIBRERIA DE TAGS</u>	21
5.1	HDIV-HTML.TLD	21
5.1.1	TAGS DE HDIV	22
5.1.2	TAGS DE STRUTS	23
5.1.3	TAG: CIPHER	23
5.2	HDIV-NESTED.TLD	23
5.3	HDIV-LOGIC.TLD	24
5.3.1	TAGS DE HDIV	24
5.3.2	TAGS DE STRUTS	24
6.	<u>LOGUER</u>	25
6.1	FORMATO DEL LOG DE HDIV	26
7.	<u>INSTALACION Y CONFIGURACION</u>	28

7.1	INSTALACIÓN	28
7.1.1	LIBRERÍAS.....	28
7.1.2	MODIFICAR EL DESCRIPTOR DE DESPLIEGUE /WEB-INF/WEB.XML	28
7.1.3	SPRING.....	30
7.2	CONFIGURACIÓN	30
7.2.1	HDIVCONFIG.XML.....	30
7.2.2	APPLICATIONCONTEXT.XML	33
7.2.3	STRUTS-CONFIG.XML.....	36
8.	EJEMPLOS	38
8.1	CONFIGURACIÓN	38
8.1.1	ESTRATEGIA.....	38
8.1.2	CONFIDENCIALIDAD	39
8.1.3	TAMAÑO MÁXIMO DEL ESTADO DE HDIV	40
8.1.4	PETICIONES DE TIPO MULTIPART	41
8.1.4.1	PARÁMETROS CONFIGURACIÓN.....	41
8.1.4.2	GESTOR DE PETICIONES MULTIPART	42
8.1.5	LOGUER	42
9.	CONCLUSIONES	43
10.	REFERENCIAS	44

1. INTRODUCCION

En la actualidad, la seguridad se ha vuelto uno de los aspectos más críticos dentro del desarrollo de los sistemas de información. Según Gartner [1] el %75 de los ataques realizados en la actualidad se realizan en las aplicaciones web, debido a la mejora de la seguridad a nivel red y sistemas operativos. En consecuencia, se considera que el %95 de las aplicaciones web son vulnerables a algún tipo de ataque [2]. En el siguiente cuadro podemos ver qué porcentaje de las aplicaciones web son vulnerables a los distintos tipos de vulnerabilidades [3]:

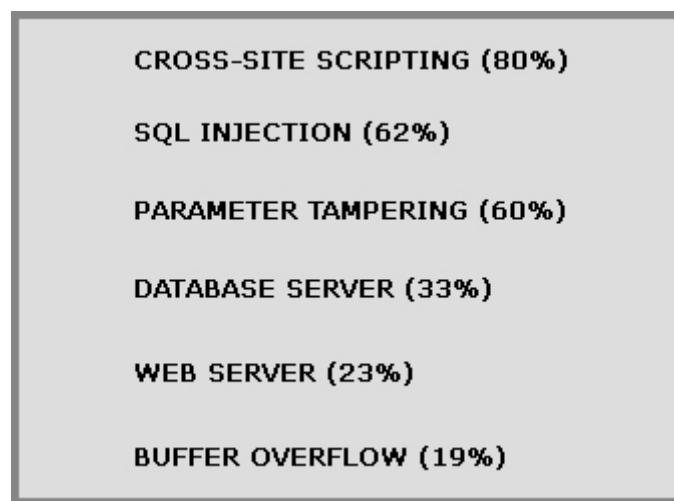


Imagen 1-1 – Porcentaje aplicaciones web vulnerables

En los siguientes puntos se describen en detalle tres de las más importantes vulnerabilidades, concretamente: parameter tampering, SQL-injection y cross-site scripting (XSS).

1.1 Parameter tampering

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.

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 (XX= *número cuenta*):

```
http://www.miBanco.com?cuenta=XX
```

Cuando el cliente acceda a la aplicación web del banco, dicha aplicación creará un link de este tipo por cada cuenta bancaria del cliente. Por tanto, pulsando dichos links, el cliente sólo podrá acceder a sus cuentas bancarias. Sin embargo, sería muy fácil para este cliente acceder a otras cuentas bancarias simplemente tecleando una nueva url en el navegador con el número de cuenta deseado.

Por esta razón, la aplicación debe verificar en el lado del servidor que el cliente tiene permiso para visualizar la cuenta que ha solicitado ver.

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

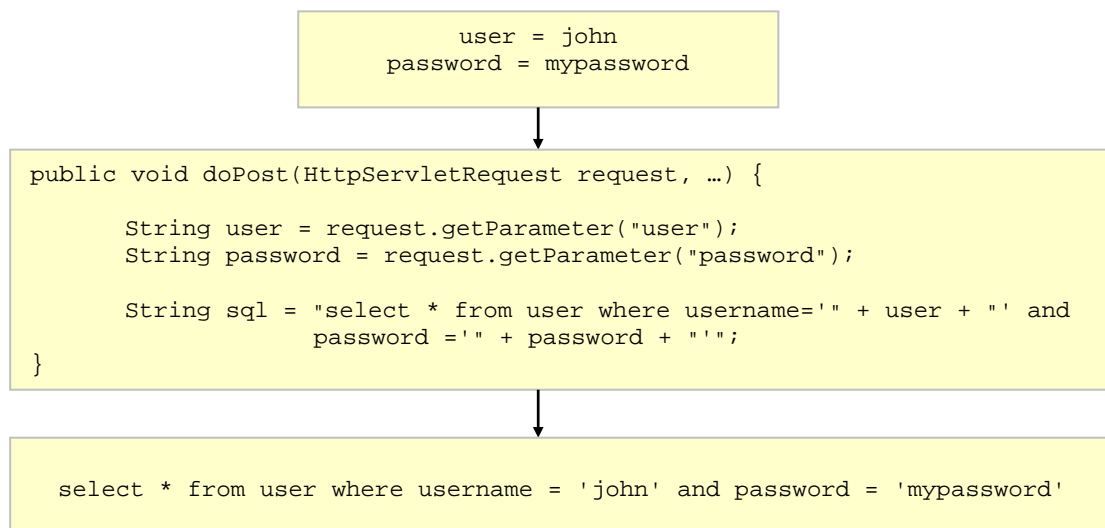
Esta vulnerabilidad está basada en la falta de verificación por parte del protocolo HTTP de los datos creados en el servidor y tiene que ser tenida en cuenta por los programadores a la hora de desarrollar nuevas aplicaciones.

Aunque en el ejemplo presentado, el elemento modificado es un link, no hay que olvidar que es posible alterar cualquier tipo de dato existente en una página web (listas seleccionables, campos ocultos, botones de radio, etc.). Es decir, la alteración de los datos no es algo inherente a las peticiones de tipo GET (links) sino que se puede realizar igualmente en peticiones de tipo POST (formularios) con el uso de herramientas de auditoria [4] utilizables por cualquier tipo de usuario que sepa manejar un navegador web.

1.2 SQL-injection

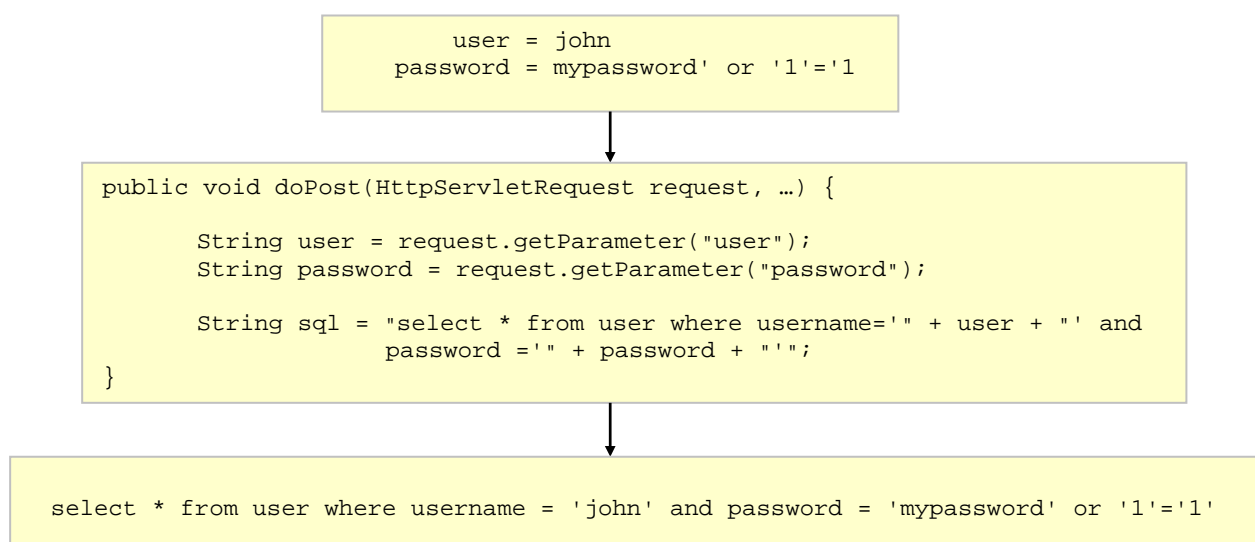
En este caso la fuente del problema radica en las malas prácticas de programación en la capa de acceso a datos.

Ejemplo: supongamos que disponemos de una página web en la que se requiere identificación al usuario mediante un formulario. Los datos requeridos en el formulario son el nombre de usuario (*user*) y el password (*password*). Una vez rellenos se envían los dos parámetros al servidor:



Como podemos apreciar en el ejemplo, la consulta SQL lanzada contra la base de datos se compone mediante la concatenación directa de los datos obtenidos desde el cliente.

Con una petición normal en la que se envían dos datos esperados, la SQL generada no comporta ningún problema, pero si los valores son modificados podemos tener un problema de seguridad. Por ejemplo, si el usuario y password enviados son los siguientes:



En este caso, la sql generada es muy diferente puesto que nos devuelve todos los usuarios de la tabla de usuarios sin haber aportado un usuario y password validos. En consecuencia, y si el programa no controla el número de registros obtenidos, podría entrar en la zona privada de la aplicación sin disponer de una credencial que lo permita.

Las consecuencias de la explotación de esta vulnerabilidad puede ser mitigada si se limitan los permisos del usuario de base de datos que es utilizado por la aplicación. Por ejemplo, si el usuario tiene permisos de borrado de datos las consecuencias de ser vulnerables a esta vulnerabilidad pueden ser extremadamente graves.

1.3 Cross-Site Scripting (XSS)

Esta técnica de ataque está basada en la inyección de código (javascript o html) en las páginas visualizadas por los usuarios de las aplicaciones.

Ejemplo: supongamos que disponemos de una página web donde podemos introducir un texto, como por ejemplo la de la siguiente imagen:

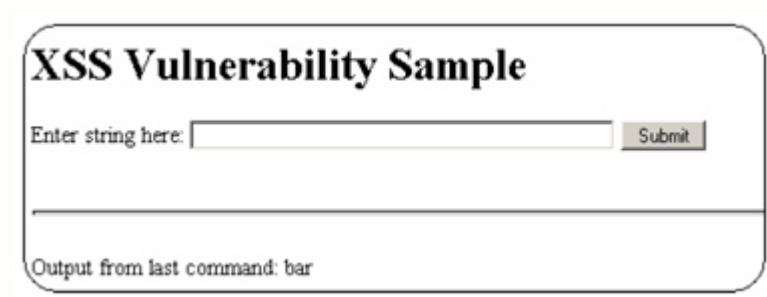


Imagen 1-2 – Ejemplo Vulnerabilidad XSS

El código HTML de la página es el siguiente:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head><title>XSS Vulnerability Sample</title></head>
<body>
<h1>XSS Vulnerability Sample</h1>

<form method="GET" action="XSS.jsp">
  Enter string here:
  <input type="text" name="userInput" size=50/>
  <input type="submit" value="Submit" />
</form>
<br><hr><br>
Output from last command: <%= request.getParameter("userInput")%>
</body>
</html>
```

Introduciendo el siguiente texto en la caja de texto:

```
<script>
    alert("If you see this you have a potential XSS vulnerability!");
</script>
```

El resultado es el siguiente:

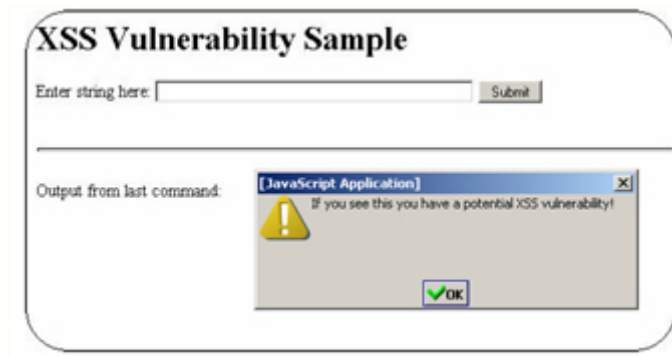


Imagen 1-3 – XSS Vulnerability Sample result

Qué puede hacer un atacante cuando nuestra aplicación es vulnerable a XSS?

El tipo de ataques que se pueden realizar son muy variados. Uno de los más conocidos es el envío de emails masivos que podemos ver en la siguiente imagen, adjuntando una url de una aplicación de confianza (en este ejemplo, happybanking) en la que el resultado final es la ejecución de una función javascript que puede por ejemplo redireccionarnos a otra web (la web del atacante con la misma imagen visual que la original) u obtener las cookies de nuestro navegador y enviarlas a la web del atacante.

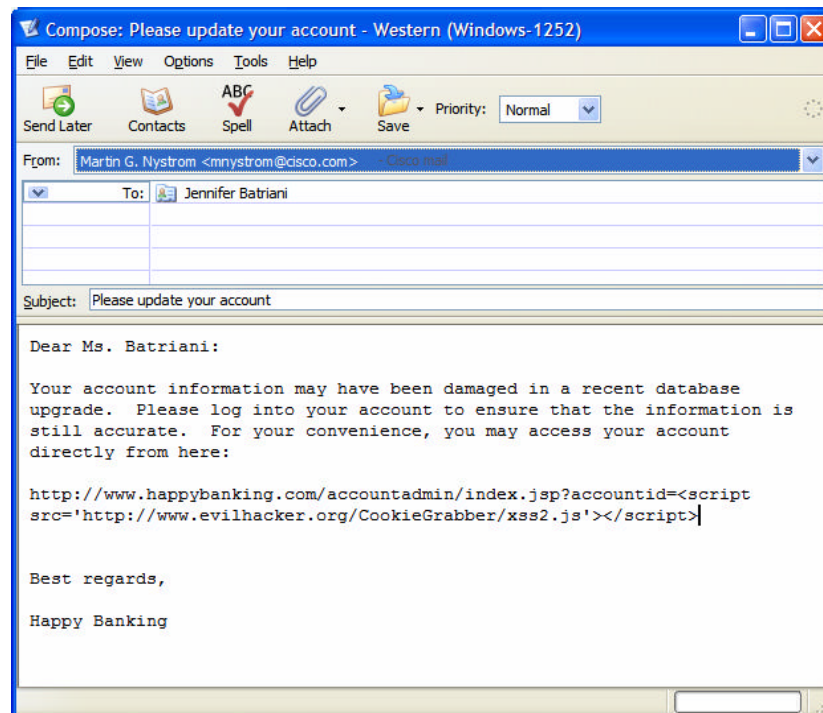


Imagen 1-4 – Ataque Mail XSS

El robo de cookies puede proporcionar al atacante el acceso a las aplicaciones web en las que el usuario esté autenticado en ese momento (banca on-line, correo personal, etc.). Esto es gracias a que la mayoría de aplicaciones web utilizan cookies como técnica de mantenimiento de la sesión. Es decir, una vez que el usuario se identifica, el servidor crea un identificador de sesión que se almacena en forma de cookie en el navegador del usuario. A partir de ahí cada petición contra la aplicación web que envía el usuario incluye esta cookie para no tener que introducir el usuario/password en cada petición. Todo esto es gestionado de forma automática por el navegador.

Esta vulnerabilidad (XSS) puede ser solucionada a través de políticas de validación genéricas (que no permiten ciertos caracteres) o utilizando librerías como Struts [5] que evitan este tipo de problemas.

2. ESTADO DEL ARTE

Todas las vulnerabilidades presentadas con anterioridad pueden ser solucionadas mediante una correcta programación de las aplicaciones web. A pesar de que se pueden solucionar, en la mayoría de los casos implica la creación de una solución propietaria y particular para cada caso.

Hay que añadir a esto que es muy probable que en los cientos de puntos en los que hay que realizar validaciones (en cada petición) los programadores olviden realizarlas en más de un punto, dejando las puertas abiertas a ataques maliciosos.

Para solucionar algunos de los problemas presentados, existen soluciones globales. Por ejemplo, el validador de Struts puede ser muy útil para solucionar vulnerabilidades de tipo *SQL-injection* o *cross-site scripting (XSS)* aunque está limitado a la comprobación de los tipos de datos. Con el validador de Struts podemos asegurar que un parámetro es de tipo entero pero no podemos saber si el valor es el mismo que el servidor envió al cliente.

La única vulnerabilidad que no es posible solucionar de forma uniforme y que es la base de la mayoría de ataques es el *parameter-tampering* o la alteración de la integridad. Controlar esta vulnerabilidad de forma manual implica un gran costo de desarrollo, además de ser muy probable la existencia de errores debido a las dificultades de probar la correcta programación de cada página.

3.

LIBRERIA DE TAGS

3.1 Introducción

Con el objeto de solucionar la mayoría de vulnerabilidades web se creó **HDIV** (HTTP Data Integrity Validator): **versión segura de Struts**, que extiende el comportamiento y añade nuevas funcionalidades de seguridad, manteniendo el API y especificaciones de Struts.

Esto implica la posibilidad de utilizar HDIV en aplicaciones desarrolladas en Struts **de forma transparente al programador** y sin añadir mayor complejidad en el desarrollo de aplicaciones. Es posible la utilización de HDIV en aplicaciones no basadas en Struts pero en este caso sí que implica la modificación de las aplicaciones (páginas JSP).

Las funcionalidades de seguridad añadidas a la versión original de Struts son las siguientes:

- ✓ **Integridad:** HDIV garantiza la integridad (la no modificación) de todos los datos generados en el servidor que no pueden ser modificados por la parte cliente (links, campos ocultos, listas seleccionables, valores de radio, páginas destino,..). Gracias a esta propiedad se eliminan todas las vulnerabilidades basadas en el parameter tampering.
- ✓ **Confidencialidad:** además de garantizar la integridad de los datos generados en el servidor HDIV garantiza la confidencialidad de todos los datos generados en el servidor. Habitualmente muchos de los datos enviados al cliente aportan información clave para los posibles atacantes como identificadores de registros de Bases de Datos, nombre de columnas o tabla, nombres de directorios web, etc.

Todos estos datos son ocultados por HDIV para evitar el uso malicioso de los mismos. Por ejemplo un link de este tipo, <http://www.host.com?dato1=12&dato2=24> es sustituido por <http://www.host.com?dato=0&dato2=1> garantizando la confidencialidad de los valores que representan identificadores de BB.DD. Además de los valores de los parámetros es posible ocultar también el nombre de los parámetros convirtiendo el link anterior en <http://www.host.com?0=0&1=1>.

3.2 Conceptos base

Antes de detallar la forma en la que HDIV garantiza la integridad y confidencialidad de los datos es necesario presentar varios conceptos base. El primero y más importante es el concepto de Estado.

Para HDIV un *estado* representa todos los datos que componen una posible petición contra una aplicación web, dicho de otro modo, los parámetros que componen una petición junto con los valores y tipos de los mismos, además del destino o página de la petición.

Por ejemplo si tenemos un link de este tipo, <http://www.host.com/pagina1.do?dato1=20&dato2=35>, el estado que representa a este link es el siguiente:

ESTADO

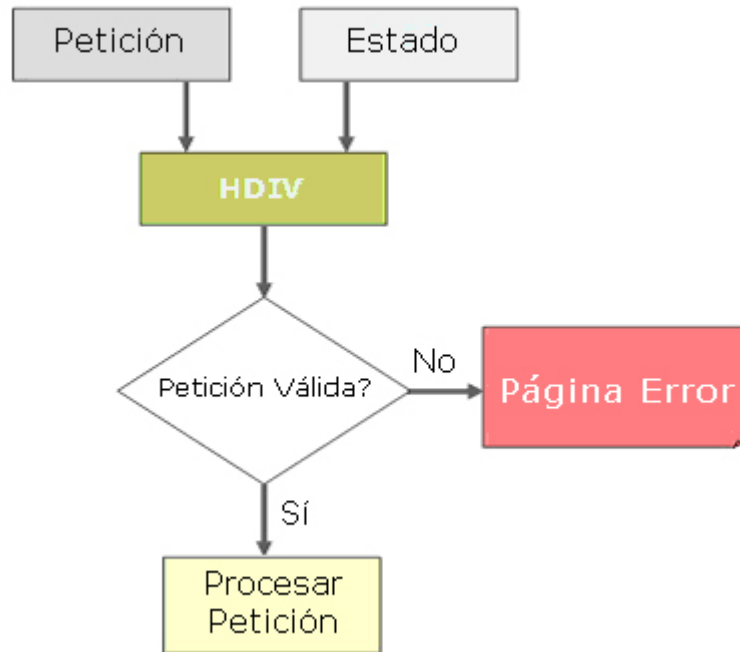
```
Action: pagina1.do
Parametros:
  dato1:
    valores: 20
    tipo: link
  dato2:
    valores: 35
    tipo: link
```

Evidentemente dentro de una página podemos tener más de un estado (posible petición) que representan a los links y formularios existentes en la página. Cuando una página (JSP) es procesada en el servidor, HDIV genera un objeto de tipo estado por cada link o formulario existente en la página (JSP).

Los estados generados pueden ser almacenados en dos localizaciones:

- **Servidor:** los estados son almacenados dentro de la sesión (HttpSession) del usuario.
- **Ciente:** los objetos estado son enviados al cliente en forma de parámetro. Es decir, a cada posible petición (link o formulario) se añade un nuevo parámetro que representa al estado de la petición.

Estos estados posibilitan la posterior verificación de las peticiones realizadas por los clientes, contrastando los datos aportados por el cliente con el estado.



3.3 Arquitectura

HDIV está compuesto por dos módulos principales:

- Librería de tags
- Filtro de Seguridad

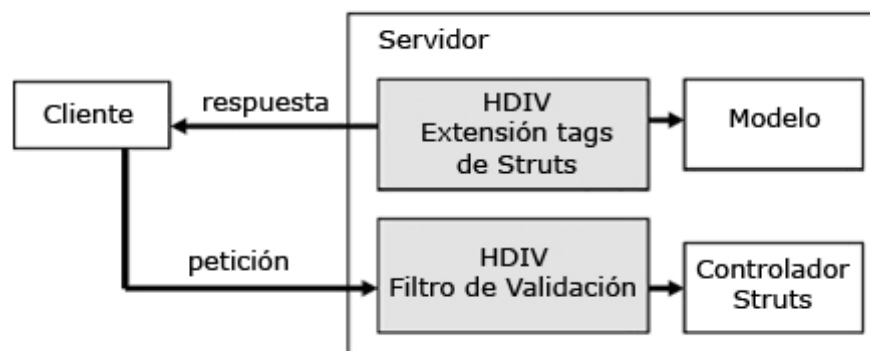


Imagen 3-1 – Arquitectura de HDIV

La librería de tags es la responsable de modificar el contenido HTML enviado al cliente para posteriormente poder ser verificado por el filtro de seguridad.

4. ESTRATEGIAS DE FUNCIONAMIENTO

Aunque los objetivos finales son los mismos, HDIV dispone de diferentes estrategias de funcionamiento:

- **Cifrado:** a cada posible petición de cada página (link o formulario) se le añade un parámetro extra (`_HDIV_STATE_`) que representa el estado de la petición.

Para garantizar la integridad del propio estado, que a la postre es la base de la validación, el estado es cifrado mediante un algoritmo de cifrado simétrico. Además de añadir el parámetro extra, todos los valores no editables son sustituidos por valores relativos (0,1,2,...) para garantizar la confidencialidad de los datos.

- **Hash:** esta estrategia de funcionamiento es muy similar a la de cifrado con la diferencia de que el estado es enviado al cliente codificado en Base64.

Para poder validar la integridad de este parámetro antes de enviar el estado se genera un hash del mismo y se almacena en la sesión para posteriormente validar la integridad del estado. Esta estrategia de funcionamiento no garantiza la confidencialidad de los datos puesto que el estado puede ser decodificado aunque se requieran de conocimientos técnicos importantes para ello.

- **Memoria:** todos los estados de cada página son almacenados en la sesión del usuario. Para poder asociar las peticiones del cliente con el estado almacenado en la sesión, se añade un parámetro extra (`_HDIV_STATE_`) a cada petición con el identificador que permite obtener el estado desde la sesión. Al igual que la estrategia de cifrado los valores no editables son ocultados al cliente garantizando su confidencialidad.

Veámos el código HTML generado por HDIV para las distintas configuraciones o estrategias así como los pasos existentes en el proceso de validación.

Supongamos que disponemos de una página que genera el siguiente código HTML donde las zonas marcadas en negrita representan a datos no editables que necesitamos proteger.

```
<html>
<body>
<a href="/struts-examples/action1.do?data=22">LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
  <input type="text" name="name" value="" />

  <input type="password" name="secret" value="" />
  <select name="color">
    <option value="10">Red</option>
    <option value="11">Green</option>
    <option value="21">Blue</option>
  </select>

  <input type="radio" name="rating" value="10" />Actually, I hate
it.<br />
  <input type="radio" name="rating" value="20" />Not so much.<br />
  <input type="radio" name="rating" value="22" />I'm indifferent<br
/>
  <textarea name="message" cols="40" rows="6"></textarea>
  <input type="hidden" name="hidden" value="15" />
  <input type="submit" value="Submit" />
</form>
</body>
</html>
```

4.1 Estrategia de Cifrado

4.1.1 Introducción

El **estado** es **enviado al cliente** en forma de campo oculto o parámetro en el caso de los links. Para garantizar la integridad, el **estado** es **cifrado** mediante un algoritmo de cifrado simétrico.

Para garantizar la confidencialidad, los **datos no editables** son **sustituídos por valores relativos**.

4.1.2 Generación de la respuesta

En primer lugar HDIV recopila toda la información que compone cada petición y genera un objeto de tipo *org.hdiv.state.IState* por cada petición existente en cada página (formularios + links). Este objeto *IState* es el que se envía al cliente en forma de objeto serializado.

La segunda responsabilidad de HDIV es la de sustituir los valores reales de los datos no editables por valores relativos. Por ejemplo si tenemos una lista seleccionable con los siguientes valores: 150, 133, 22 los sustituye por valores relativos: 0, 1, 2.

De esta forma se garantiza la confidencialidad de los datos no editables. Una vez generado el objeto *IState*, éste será enviado al cliente en un parámetro hidden para los formularios o como parámetro extra en los links.

Los pasos a realizar para calcular el valor de este parámetro son los siguientes:

- Se obtiene el array de bytes del objeto *IState* (éste tiene que ser serializable).
- Se comprime.
- Se encripta.
- Se codifica a Base64.

El resultado de una página con la estrategia de cifrado y activación de la confidencialidad tendrá este aspecto:

```
<html>
<body>
<a href="/struts-
examples/action1.do?data=0&_HDIV_STATE=6347dfhdfd84r73e9483494734837487">
LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
  <input type="text" name="name" value="" />
  <input type="password" name="secret" value="" />
  <select name="color">
    <option value="0">Red</option>
    <option value="1">Green</option>
    <option value="2">Blue</option>
  </select>

  <input type="radio" name="rating" value="0" />Actually, I hate
it.<br />
  <input type="radio" name="rating" value="1" />Not so much.<br />
  <input type="radio" name="rating" value="2" />I'm indifferent<br
/>
  <textarea name="message" cols="40" rows="6"></textarea>
  <input type="hidden" name="hidden" value="0" />
  <input type="hidden" name="_HDIV_STATE_"
    value="jkhdfhgdf948dkfhdfdkhffjfdf" />
  <input type="submit" value="Submit" />
</form>
</body>
</html>
```

4.1.3 Validación

El primer paso en el proceso de validación será descifrar el valor del parámetro `_HDIV_STATE_`, que contiene el estado de la petición.

Si no hay errores al descifrar el estado, el valor no ha sido modificado y por tanto podemos seguir procesando la petición.

El siguiente paso es descomprimir el valor del parámetro y volver a crear un objeto de tipo *IState* a partir de los bytes obtenidos.

Con el objeto *IState* restaurado estamos preparados para validar la petición del cliente. Para ello, recorreremos cada uno de los parámetros que contiene la petición y a su vez cada uno de los valores de cada parámetro.

```
While (parametros) {  
    While (valores)  
    {  
        DataValidator.validate(valor);  
    }  
}
```

HDIV verifica en primer lugar que el valor del parámetro está dentro de los posibles valores relativos del parámetro.

Si es correcto devuelve el valor real de la opción seleccionada por el cliente. Es decir, si por ejemplo el valor del parámetro cuenta es 0, HDIV sustituye ese valor por el valor existente dentro de la lista de valores del parámetro, concretamente el situado en la posición 0.

Si la petición es correcta es redirigida al controlador de Struts para que genere la página que corresponda. En caso de existir algún tipo de problema de seguridad se genera un log con la incidencia y se redirige al usuario a una página de error.

4.2 Estrategia Hash

4.2.1 Introducción

El **estado** es **codificado en Base64** y **enviado al cliente** en forma de campo oculto o parámetro en el caso de los links.

Para garantizar la integridad, antes de enviar el estado al cliente se genera el hash de éste y se almacena en la sesión del usuario para posteriormente verificar la no alteración del valor.

La principal diferencia de esta versión respecto a la versión anterior es que en vez de garantizar la integridad del estado a través del cifrado se realiza mediante el uso de un hash.

Cabe destacar que en este caso no es posible garantizar la confidencialidad de los datos en cliente al no ir cifrados los datos. De todas formas, los datos reales no se envían dentro de cada componente con el fin de que el proceso de conocer los valores reales no sea tan sencillo.

4.2.2 Generación de la respuesta

Visualmente el resultado de una página con esta estrategia es el mismo que la versión anterior.

```
<html>
<body>
<a href=/struts-
examples/action1.do?data=0&_HDIV_STATE=wJTAWJTAbVALQzFKJUMzJTQwJTE>
LinkRequest</a>

<form method="post" action="/struts-examples/processSimple.do">
  <input type="text" name="name" value="" />
  <input type="password" name="secret" value="" />
  <select name="color">
    <option value="0">Red</option>
    <option value="1">Green</option>
    <option value="2">Blue</option>
  </select>

  <input type="radio" name="rating" value="0" />Actually, I hate
it.<br />
  <input type="radio" name="rating" value="1" />Not so much.<br />
  <input type="radio" name="rating" value="2" />I'm indifferent<br
/>

  <textarea name="message" cols="40" rows="3"></textarea>
  <input type="hidden" name="hidden" value="0" />
  <input type="hidden" name="_HDIV_STATE_"
value="lRUMlQCuwMOYlMUyLRMUCwMRQlQzFj" />

  <input type="submit" value="Submit" />
</form>
</body>
</html>
```

4.2.3 Validación

La única diferencia respecto a la versión anterior es que en este caso, se sustituye el proceso de descifrar los datos por el de la verificación de la integridad utilizando el hash. Para verificar la integridad se genera el hash del valor que representa al estado y se compara con el almacenado en sesión.

A partir de ahí la verificación de la petición es exactamente igual que en el caso anterior.

4.3 Estrategia de Memoria

4.3.1 Introducción

El **estado** de cada petición se almacena **en la sesión del usuario**, siendo la principal diferencia con respecto a las anteriores versiones comentadas.

Para garantizar la confidencialidad, los **datos no editables son sustituidos por valores relativos**.

4.3.2 Generación de la respuesta

La diferencia con respecto a las versiones comentadas más arriba es que en este caso el estado no es enviado al cliente, solamente se envía el identificador de la petición para poder recuperar el estado de la petición con posterioridad.

```
<html>
<body>
<a href="/struts-examples/action1.do?data=0&_HDIV_STATE=0-1">
LinkRequest</a>
<form method="post" action="/struts-examples/processSimple.do">
  <input type="text" name="name" value=""/>
  <input type="password" name="secret" value="" />
  <select name="color">
    <option value="0">Red</option>
    <option value="1">Green</option>
    <option value="2">Blue</option>
  </select>
  <input type="radio" name="rating" value="0" />Actually, I hate
it.<br />
  <input type="radio" name="rating" value="1" />Not so much.<br />
  <input type="radio" name="rating" value="2" />I'm indifferent<br />
  <textarea name="message" cols="40" rows="3"></textarea>
  <input type="hidden" name="hidden" value="0" />
  <input type="hidden" name="_HDIV_STATE_" value="0-2" />
  <input type="submit" value="Submit" />
</form>
</body>
</html>
```

Como se puede apreciar visualmente no hay diferencias entre la versión de estado en cliente o en servidor, la única diferencia es la longitud del valor del parámetro que representa al estado (`_HDIV_STATE_`) que en el caso de la versión basada en servidor es bastante más pequeña puesto que únicamente contiene el identificador de la petición.

4.3.3 Validación

Antes de iniciar la validación, se obtiene el identificador de la petición enviado al cliente y se recupera el objeto de tipo *org.hdiv.state.IState* de la sesión del usuario para realizar la validación.

A partir de ahí la validación es exactamente igual que en las versiones anteriores.

5. LIBRERÍA DE TAGS

HDIV dispone de tres librerías de tags:

- *hdiv-html.tld*: extiende el comportamiento y sintaxis los tags HTML de Struts.
- *hdiv-nested.tld*: extiende el comportamiento y sintaxis de los tags Nested de Struts.
- *hdiv-logic.tld*: extiende el comportamiento y sintaxis de los tags Logic de Struts.

Como se ha comentado en el capítulo 2.1, HDIV es una versión segura de Struts. que **extiende el comportamiento** y añade nuevas funcionalidades de seguridad, **manteniendo el API y especificaciones de Struts**.

Existen por tanto diferentes versiones de las librerías *hdiv-html.tld*, *hdiv-nested.tld* y *hdiv-logic.tld* compatibles con las diferentes versiones de Struts. Actualmente, HDIV dispone de librerías de tags compatibles con las siguientes versiones de Struts:

Versión de Struts	Librería
1.1	<i>hdiv-struts-1.1.jar</i>
1.2.4	<i>hdiv-struts-1.2.4.jar</i>
1.2.7	<i>hdiv-struts-1.2.7.jar</i>
1.2.9	<i>hdiv-struts-1.2.9.jar</i>

5.1 hdiv-html.tld

La librería *hdiv-html.tld* extiende el comportamiento y sintaxis de los tags HTML de Struts habiendo modificado la implementación de algunos tags (*ver apartado 5.1.1*) para conseguir la integridad y confidencialidad de los datos en HDIV.

“The tags in the Struts HTML library form a bridge between a JSP view and the other components of a Web application. Since a dynamic Web application often depends on gathering data from a user, input forms play an important role in the Struts framework. Consequently, the majority of the HTML tags involve HTML forms.”

The HTML taglib contains tags used to create Struts input forms, as well as other tags generally useful in the creation of HTML-based user interfaces. The output is HTML 4.01 compliant or XHTML 1.0 when in XHTML mode."

El resultado HTML obtenido por los tags de la librería *hdiv-html.tld* es el mismo que el que se obtendría con el TLD *struts-html.tld* salvo algunas excepciones comentadas en el apartado 5.1.1.

Por tanto, al haber respetado la sintaxis y/o definición de los tags HTML de Struts, las páginas JSPs no deben de ser re-programadas al utilizar HDIV.

5.1.1 Tags de HDIV

Los únicos tags que generan HTML diferente al que se obtiene con la librería *struts-html.tld* son los tags *form* y *link*.

En el tag *form* se añade un nuevo campo oculto con el estado de HDIV (ver apartado 3.2). En cambio, en el tag *link* el estado de HDIV se añade como un nuevo parámetro.

- *button*: render a button input field.
- *cancel*: render a cancel button.
- *checkbox*: render a checkbox input field.
- *file*: render a file input field.
- *form*: render an input form.
- *hidden*: render a hidden field.
- *link*: render an HTML anchor or hyperlink.
- *multibox*: render a checkbox input field
- *option*: render a select option
- *options*: render a collection of select options
- *optionsCollection*: render a collection of select options
- *password*: render a password input field
- *radio*: render a radio button input field
- *select*: render a select element
- *submit*: render a submit button
- *text*: render an input field of type text
- *textarea*: render a textarea

Cuando el proceso de confidencialidad está activado, el valor obtenido en la propiedad *value* de los tags *hidden*, *multibox*, *optionsCollection*, *options*, *option* y *radio* es un valor codificado únicamente entendible por HDIV.

La definición y propiedades de los tags podemos consultarla en la guía de usuario de Struts [5]. Tenga en cuenta que en función de su versión de Struts deberá consultar su correspondiente guía de usuario ya que la especificación de los tags en cada versión es diferente.

5.1.2 Tags de Struts

El resto de tags definidos en la librería *hdiv-html.tld* (*base*, *errors*, *frame*, *html*, *javascript*, *reset*, *rewrite*, *xhtml* y *messages*) no han sido modificados por HDIV y por tanto son los tags de Struts con el mismo nombre los que se utilizan en HDIV.

5.1.3 Tag: cipher

HDIV aporta un nuevo tag *cipher* con el que se podrá cifrar cualquier valor deseado siempre y cuando a dicho tag se le indiquen valores para las siguientes propiedades: *property*, *action* y *value*. Veámos un ejemplo:

Supongamos que deseamos cifrar un valor de un campo *hidden* sin utilizar el tag *hidden* proporcionado por la librería *hdiv-html.tld*.

```
<%@ taglib uri="/WEB-INF/hdiv-html.tld" prefix="html" %>
<input type="hidden" value="<hdiv:cipher action="a1" property="p1"
value="v1" />" />
```

El resultado obtenido sería el siguiente:

```
<%@ taglib uri="/WEB-INF/hdiv-html.tld" prefix="html" %>
<input type="hidden" value="0" />
```

5.2 hdiv-nested.tld

La librería *hdiv-nested.tld* extiende el comportamiento y sintaxis de los tags *Nested* de Struts.

"This tag library brings a nested context to the functionality of the Struts custom tag library."

It's written in a layer that extends the current Struts tags, building on their logic and functionality. The layer enables the tags to be aware of the tags which surround them so they can correctly provide the nesting property reference to the Struts system."

El resultado HTML obtenido por los tags de la librería *hdiv-nested.tld* es el mismo que el que se obtendría con el TLD *struts-nested.tld* salvo en los tags *form* y *link* en los que se añade el estado de HDIV al igual que en los tags *form* y *link* de la librería *hdiv-html.tld*.

Cuando el proceso de confidencialidad está activado, el valor obtenido en la propiedad *value* de los tags *hidden*, *multibox*, *optionsCollection*, *options* y *radio* es un valor codificado únicamente entendible por HDIV.

La definición y propiedades de los tags podemos consultarla en la guía de usuario de Struts [5].

5.3 hdiv-logic.tld

La librería *hdiv-logic.tld* extiende el comportamiento y sintaxis de los tags LOGIC de Struts (ver apartado 5.3.1) para conseguir la integridad y confidencialidad de los datos en HDIV.

Por tanto, al haber respetado la sintaxis y/o definición de los tags LOGIC de Struts, las páginas JSPs no deben de ser re-programadas al utilizar HDIV.

5.3.1 Tags de HDIV

En HDIV los dos únicos tags que se han extendido de la librería LOGIC son: *forward* y *redirect*.

- forward
- redirect

5.3.2 Tags de Struts

El resto de tags definidos en la librería *hdiv-logic.tld* (*empty*, *equal*, *greaterEqual*, *greaterThan*, *iterate*, *lessEqual*, *lessThan*, *match*, *messagesNotPresent*, *messagesPresent*, *notEmpty*, *notEqual*, *notMatch*, *notPresent*, *present*) no han sido modificados por HDIV y por tanto son los tags de Struts con el mismo nombre los que se utilizan en HDIV.

6. LOGUER

HDIV dispone de un logger que imprimirá todos los ataques detectados por HDIV consiguiendo así que los administradores de sistemas puedan consultar los ataques realizados sobre la aplicación web.

HDIV utiliza la API de Commons Logging [8], que a su vez puede utilizar Log4j [9] como mecanismo de logging subyacente. Si la biblioteca Log4j está disponible en el directorio de bibliotecas del contexto, Commons Logging hará uso de Log4j y de la configuración definida en el fichero *log4j.properties* situado en el classpath de contexto.

Se distribuye con HDIV un fichero de propiedades de ejemplo (*log4j.properties*) para Log4j situado en el directorio: */hdiv-web-struts-1.1/src/main/resources/*. En la configuración proporcionada podemos loguear los ataques detectados por HDIV en un fichero de logging:

```
#
# Configuration for a rolling log file ("hdiv.log").
#
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.DatePattern='.'yyyy-MM-dd
#
# Edit the next line to point to your logs directory.
# The last part of the name is the log file name.
#
log4j.appender.R.File=C://hdiv.log
log4j.appender.R.layout=org.apache.log4j.PatternLayout
#
# Print the date in ISO 8601 format
#
log4j.appender.R.layout.ConversionPattern=%d [%t] %-5p - %m%n
#
# Application logging options
#
log4j.logger.org.hdiv=INFO,R
```

Mediante la configuración mostrada los mensajes de log se escribirán en el fichero situado en *c://hdiv.log* con el formato definido en la propiedad *log4j.appender.R.layout.ConversionPattern* donde:

- %d : Fecha del evento de log
- [%t]: nombre del thread que generó el evento de log
- %-5p: prioridad del evento que generó el logging
- %m: mensaje asociado con el evento de logging
- %n: carácter de salto de línea

Para diferentes configuraciones del mensaje de log podemos consultar la documentación de Log4j [9].

6.1 Formato del log de HDIV

El mensaje de log generado por HDIV, debido al carácter de conversión *%m* definido en el patrón de conversión declarado en *log4j.properties*, tiene el siguiente formato:

```
[tipo ataque];[accion];[parametro];[valor];[IPlocalusuario];[IP];[idUsuario]
```

[tipo ataque]: Tipo de ataque detectado por HDIV. Los posibles valores son:

- 1: La acción o destino recibido en la petición no corresponde con el destino almacenado en el estado
- 2: El parámetro recibido en la petición no existe en el estado de la petición
- 3: No se han recibido todos los parámetros requeridos
- 4: Valor incorrecto del parámetro
- 5: No se han recibido todos los valores esperados para el parámetro
- 6: Se han recibido valores repetidos para un mismo parámetro
- 7: Valor incorrecto. Confidencialidad activada.
- 8: No se ha recibido el parámetro de HDIV en la petición
- 9: El parámetro de HDIV tiene un valor incorrecto
- 10: El parámetro de HDIV tiene un identificador de página incorrecto

[accion]: url o acción contra la que va dirigida la petición HTTP.

[parametro]: parámetro de la petición HTTP.

[valor]: valor del parámetro *[parametro]*.

[IPlocalusuario]: Dirección IP si la petición se ha realizado a través de un proxy.

[IP]: Dirección IP desde donde se ha realizado la petición.

[idUsuario]: Identificador de usuario. La manera en que cada aplicación web obtiene el identificador de usuario puede variar, es por ello por lo que se ha definido un interfaz (*IUserData*) posibilitando implementar diferentes formas para la obtención de la identidad del usuario.

Veámos algunos ataques detectados por HDIV:

```
2006-09-22 10:56:07,214 [http-80-Processor25] INFO -  
1;action1;param1;value1;188.15.1.25;201.166.24.12;45652146M  
2006-09-22 10:58:15,500 [http-80-Processor25] INFO -  
7;action3;param2;value3;188.15.1.25;201.166.24.12;15235687G  
2006-09-22 11:01:24,124 [http-80-Processor25] INFO -  
3;action5;param1;value1;188.15.1.25;201.166.24.12;15235687G  
2006-09-22 11:15:00,411 [http-80-Processor25] INFO -  
2;action1;param5;value2;188.15.1.25;201.166.24.12;45652146M
```

7. INSTALACION Y CONFIGURACION

Prácticamente toda la configuración está definida mediante Spring, dejando en el descriptor de despliegue la definición del filtro de HDIV y los listeners de Spring y HDIV.

A continuación se describen los pasos a realizar para la instalación y configuración de HDIV en una aplicación web.

7.1 Instalación

Los pasos a realizar en la instalación de HDIV son los siguientes:

7.1.1 Librerías

- Incluir las siguientes librerías en el classpath de la aplicación web (ya sea dentro de WEB-INF/lib o en el classpath de nivel servidor):
 - HDIV: librería core (*hdiv-core.jar*) y librería de tags (existen diferentes versiones para las diferentes versiones de Struts. *Ver capítulo 5*)
 - Spring: *spring-1.2.6.jar*
 - Commons codec: *commons-codec-1.3.jar*
- Librerías de tags: añadir las siguientes librerías en el directorio WEB-INF de nuestra aplicación web.
 - *hdiv-html.tld*
 - *hdiv-nested.tld*
 - *hdiv-logic.tld*

7.1.2 Modificar el descriptor de despliegue /WEB-INF/web.xml

La configuración a añadir al descriptor de nuestra aplicación es la siguiente:

- Localización del fichero de configuración de Spring:

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    /WEB-INF/applicationContext.xml , /WEB-INF/hdivConfig.xml
  </param-value>
</context-param>
```

- Añadir los listeners de inicialización de HDIV y Spring.

- o Listener de Spring:

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

- o Listener de HDIV:

- Entorno WebSphere:

```
<listener>
  <listener-class>
    org.hdiv.listener.InitWebSphereListener
  </listener-class>
</listener>
```

- Entorno diferente a WebSphere:

```
<listener>
  <listener-class>
    org.hdiv.listener.InitListener
  </listener-class>
</listener>
```

- Filtro de Validación.

```
<filter>
  <filter-name>ValidatorFilter</filter-name>
  <filter-class>org.hdiv.filter.ValidatorFilter</filter-
class>
</filter>

<filter-mapping>
  <filter-name>ValidatorFilter</filter-name>
  <url-pattern>*.do</url-pattern>
</filter-mapping>

<filter-mapping>
  <filter-name>ValidatorFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

- Sustitución de las referencias a los TLDs HTML, Nested y Logic de Struts: modificar las referencias de las librerías HTML, Nested y Logic de Struts para que hagan referencia a las de HDIV.

```
<taglib>
  <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
  <taglib-location>/WEB-INF/hdiv-html.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
  <taglib-location>/WEB-INF/hdiv-nested.tld</taglib-location>
</taglib>
<taglib>
  <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
  <taglib-location>/WEB-INF/hdiv-logic.tld</taglib-location>
</taglib>
```

7.1.3 Spring

Añadir los ficheros *applicationContext.xml* y *hdivConfig.xml* en el directorio *WEB-INF* de nuestra aplicación web.

7.2 Configuración

La configuración de HDIV está definida en dos ficheros que serán consumidos por Spring [6]:

- *hdivconfig.xml*: configuración básica de HDIV para usuarios con pocos conocimientos.
- *applicationContext.xml*: definición de beans de Spring configurables por usuarios con avanzados conocimientos en aplicaciones web.

7.2.1 hdivConfig.xml

- Estrategia de HDIV: indicar en el tag *value* definido en el bean *strategy* el tipo de estrategia a utilizar por HDIV para la integridad de datos. Los posibles valores son: **memory**, **cipher** o **hash**.

```
<bean id="strategy" class="java.lang.String">
  <constructor-arg>
    <value>estrategia definida por el usuario</value>
  </constructor-arg>
</bean>
```

Por ejemplo, para la estrategia de memoria la configuración será la siguiente:

```
<bean id="strategy" class="java.lang.String">
  <constructor-arg>
    <value>memory</value>
  </constructor-arg>
</bean>
```

- Confidencialidad: indicar en el tag *value* el valor **true** para la activación de la confidencialidad, o **false** para su desactivación.

```
<bean id="confidentiality" class="java.lang.Boolean">
  <constructor-arg>
    <value>configuración definida por el usuario</value>
  </constructor-arg>
</bean>
```

Por ejemplo, para la activación de la confidencialidad la configuración será la siguiente:

```
<bean id="confidentiality" class="java.lang.Boolean">
  <constructor-arg>
    <value>true</value>
  </constructor-arg>
</bean>
```

- Parámetros de inicio: Parámetros configurables por el usuario para la inicialización de HDIV en el bean config.

```
<bean id="config" class="org.hdiv.config.HDIVConfig">
```

- Página de error: definir la ruta del JSP al que será redireccionado la petición cuando no supere la validación.

```
<property name="errorPage">
  <value>/jsps/error.jsp</value>
</property>
```

- Páginas de inicio: por defecto HDIV sólo acepta peticiones a destinos que han sido enviados anteriormente al cliente en el código HTML de las páginas. Si se intenta acceder a una página directamente mediante el navegador, HDIV redirigirá la petición a la página de error.

Todas las aplicaciones web tienen una página de inicio o "home page" a la cuál los usuarios pueden acceder directamente. Estas páginas son conocidas como "páginas de inicio" en HDIV y hace falta definir las en la configuración de HDIV. Por ejemplo, si la url de la página de inicio de la aplicación web es <http://www.host.com/nombre-aplicacion?inicio.do>, la configuración sería la siguiente:

```
<property name="userStartPages">
  <list>
    <value>/nombre-aplicacion/inicio.do</value>
  </list>
</property>
```

- Parámetros de inicio: Como se ha comentado anteriormente, HDIV asegura la integridad de los datos y la confidencialidad de los mismo. Por tanto, si se realiza una petición con un nuevo parámetro no existente en la página HTML devuelta por el servidor, dicha petición será redireccionada a la página de error.

En algunos casos, Struts genera nuevos parámetros para asegurar por ejemplo que un mismo formulario no sea enviado varias veces. Para evitar este problema Struts crea automáticamente parámetros (*org.apache.struts.taglib.html.TOKEN*, *org.apache.struts.action.TOKEN*) que no son generados por el programador. Esto implica que dichos parámetros tengan que ser definidos como "parámetros de inicio" consiguiendo así que peticiones con dichos parámetros no sean paradas por HDIV.

La configuración a definir sería la siguiente:

```
<property name="userStartParameters">
  <list>
    <value>org.apache.struts.action.TOKEN</value>
    <value>org.apache.struts.taglib.html.TOKEN</value>
  </list>
</property>
```

- Parámetros que no requieren validación: es posible definir parámetros asociados a un action o destino con el objetivo de que HDIV no los tenga en cuenta en el proceso de validación, no comprobando la integridad de los mismos. Un caso en el que puede resultar interesante la definición de estos parámetros es en funciones javascript.

Supongamos que queremos añadir los parámetros *param1* y *param2* asociados al action *action1*, y el parámetro *param1* asociado al action *action2*. La configuración sería la siguiente:

```
<property name="paramsWithoutValidation">
  <map>
    <entry key="/nombre-aplicacion/action1.do">
      <list>
        <value>param1</value>
        <value>param2</value>
      </list>
    </entry>
    <entry key="/ nombre-aplicacion /action2.do">
      <list>
        <value>param1</value>
      </list>
    </entry>
  </map>
</property>
```

Como podemos observar, por cada parámetro debemos añadir el action relacionado en el tag *entry* y el nombre del parámetro en el tag *value* como parte del tag *list*.

Aunque existen otros dos parámetros definidos en el bean *config*, no vamos a tratarlos ya que no deberían ser modificados por el usuario. Por tanto, damos por finalizado la configuración del bean *config*.

```
</bean>
```

7.2.2 applicationContext.xml

- Tamaño máximo de caché: el usuario puede definir el número máximo de estados cacheables en memoria.

Cuando el tamaño máximo de la caché se supera, los estados que más tiempo llevan almacenados en memoria se eliminan siendo reemplazados por los estados de las últimas páginas visitadas, lo que puede provocar que las páginas que sean accedidas mediante el botón "Atrás" del navegador, dejen de funcionar. Por tanto, es aconsejable definir un valor elevado para aplicaciones en donde el botón "Atrás" del navegador va a ser utilizado frecuentemente.

Aun siendo este bean utilizado por las tres estrategias disponibles en HDIV, es en las estrategias de Memoria y Hash donde más importancia tiene ya que los estados siempre se almacenan en la sesión del usuario.

Hay que tener en cuenta que un consumo elevado de memoria por cada usuario puede impactar directamente en el rendimiento general de la aplicación siendo por

tanto realmente importante medir el rendimiento de la aplicación web con diferentes valores máximos de caché.

```
<beanid="cache"class="org.hdiv.session.StateCache"  
singleton="false"init-method="init">  
  <propertyname="maxSize">  
    <value>500</value>  
  </property>  
</bean>
```

- Tamaño máximo de caracteres: existe una limitación en las peticiones HTTP de tipo GET que no permite peticiones superiores a un determinado número de bytes. Esta cantidad de bytes limita por tanto el tamaño del estado de HDIV que se almacena en la parte cliente en las estrategias de Cifrado y Hash.

Con el fin de evitar este problema, HDIV dispone de una propiedad (*allowedLength*) configurable por el usuario con la que se indica el tamaño máximo de caracteres que puede ocupar un estado para ser guardado en la parte cliente. Si el estado supera ese número de caracteres será almacenado en memoria haciendo uso de la sesión de usuario.

Veámos en el fichero de configuración *applicationContext.xml*, tanto para la estrategia de memoria como para la de hash, esta propiedad configurable.

```
<!--CIPHER STRATEGY -->
<bean id="dataComposerCipher"
      class="org.hdiv.dataComposer.DataComposerCipher"
      singleton="false" init-method="init">

    <propertyname="application">
        <refbean="application"/>
    </property>
    <propertyname="page">
        <refbean="page"/>
    </property>
    <propertyname="encodingUtil">
        <refbean="encoding"/>
    </property>
    <propertyname="allowedLength">
        <value>2000</value>
    </property>
    <propertyname="confidentiality">
        <refbean="confidentiality"/>
    </property>
</bean>

<!--HASH STRATEGY -->
<bean id="dataComposerHash"
      class="org.hdiv.dataComposer.DataComposerHash"
      singleton="false" init-method="init">

    <propertyname="application">
        <refbean="application"/>
    </property>
    <propertyname="page">
        <refbean="page"/>
    </property>
    <propertyname="encodingUtil">
        <refbean="encoding"/>
    </property>
    <propertyname="allowedLength">
        <value>2000</value>
    </property>
    <propertyname="confidentiality">
        <refbean="confidentiality"/>
    </property>
</bean>
```

- Configuración de peticiones de tipo Multipart: es necesario configurar los parámetros que se comentan a continuación para las peticiones de tipo multipart.
 - maxFileSize: tamaño máximo (en bytes) permitido de un fichero para que sea aceptado en el proceso de carga. Puede expresarse como un número seguido por "K", "M", o "G", los cuáles significan kilobytes, megabytes, o gigabytes, respectivamente. El valor por defecto es 250M.

- o memFileSize: Tamaño máximo (en bytes) del fichero que será cargado en memoria antes de realizar el proceso de carga. Los ficheros que superen este tamaño máximo serán guardados en otro destino, normalmente en el disco duro. Puede expresarse como un número seguido por "K", "M", o "G", los cuáles significan kilobytes, megabytes, o gigabytes, respectivamente. El valor por defecto es 256K.
- o tempDir: directorio temporal a utilizar en el proceso de carga de ficheros.

```
<bean id="multipartConfig"
      class="org.hdiv.config.MultipartConfig">
    <property name="maxFileSize">
      <value>250M</value>
    </property>
    <property name="memFileSize">
      <value>256K</value>
    </property>
    <property name="tempDir">
      <value>c:\tmp</value>
    </property>
  </bean>
```

7.2.3 struts-config.xml

En caso de que nuestra aplicación web realice peticiones de tipo multipart, debemos configurar el controlador de Struts para que haga uso del gestor de HDIV en este tipo de peticiones. Para ello, debemos añadir la propiedad *multipartClass* al controlador de Struts en el fichero de configuración *struts-config.xml* con el siguiente valor:

```
<controller multipartClass="org.hdiv.upload.HDIVMultipartRequestHandler"
/>
```

Con esta configuración conseguiremos que las peticiones de tipo multipart hagan uso de HDIV tomando como parámetros de configuración los definidos en el *capítulo 7.2.2*.

En caso de que nuestra aplicación web disponga de varios módulos, deberemos modificar el fichero de configuración *struts-config.xml* perteneciente al módulo donde se encuentra el upload de los ficheros.

IMPORTANTE: Antes de aplicar HDIV a nuestra aplicación web, debemos **eliminar manualmente los JSPs compilados en el servidor**, asegurándonos así que no existen JSPs de ejecuciones anteriores realizadas en nuestra aplicación sin HDIV.

A continuación se presentan los directorios donde llevar a cabo la eliminación de los JSPs compilados tanto en Tomcat como en Websphere suponiendo que nuestra aplicación es *example-app* y el nombre del servidor es *localhost*:

- Tomcat:

C:\<tomcat-installation-dir>\work\Catalina\localhost\app\

- Websphere:

- v5.1:

C:\<websphere-installation-dir>\wsappdev51\workspace\metadata\plugins\com.ibm.etools.server.core\tmp0\cache\domain\server1\app\app.war

- v6.0.1:

C:\<websphere-installation-dir>\runtimes\base_v6\profiles\default\temp\domain\server1\appEAR\appWEB.war

8. EJEMPLOS

En la distribución de HDIV nos encontramos con una aplicación de ejemplo (*struts-examples*) con la que mediante diferentes configuraciones podremos aplicar las diferentes estrategias disponibles en HDIV.

A continuación se detallan los pasos a seguir para su descarga, instalación y configuración:

1. Descargar la aplicación *struts-examples* en la sección ejemplos de la página principal de HDIV: <http://www.hdiv.org>. En función de nuestro servidor web deberemos descargar un archivo u otro (*war* o *ear*).
2. Desplegar el archivo descargado en nuestro servidor web.

En algunos servidores de aplicaciones J2EE como Tomcat debemos copiar *struts-examples.war* en el directorio *webapps* del directorio de instalación del servidor. En cambio en otros como IBM WebSphere debemos añadir *struts-examples.ear* como una nueva aplicación mediante la Consola Administrativa.

3. Arrancar el servidor web.
4. Ejecutar la URL de inicio de la aplicación *struts-examples*. Para ello deberemos conocer el dominio y el puerto sobre el que está el servidor web corriendo: <http://<domain:port>/struts-examples/>

8.1 Configuración

Por defecto *struts-examples* viene definida con la estrategia de Memoria y el proceso de Confidencialidad activado. Podremos modificar dicha configuración de la siguiente forma:

8.1.1 Estrategia

Como hemos visto en el capítulo 4 - *Operation Strategy* – existen tres estrategias en HDIV: memoria, cifrado y hash. El **único paso** a realizar para el **cambio de estrategia** es el siguiente:

1. Configurar en el fichero *hdivConfig.xml* el bean *strategy*. Los posibles valores son: memory, cipher o hash.
 - a. Estrategia de Memoria (configuración por defecto en *struts-examples*):

```
<bean id="strategy" class="java.lang.String">
  <constructor-arg>
    <value>memory</value>
  </constructor-arg>
</bean>
```

b. Estrategia de Cifrado:

```
<bean id="strategy" class="java.lang.String">
  <constructor-arg>
    <value>cipher</value>
  </constructor-arg>
</bean>
```

c. Estrategia Hash:

```
<bean id="strategy" class="java.lang.String">
  <constructor-arg>
    <value>hash</value>
  </constructor-arg>
</bean>
```

8.1.2 Confidencialidad

Como hemos visto en el capítulo 3.1 – *HDIV Introduction* – es posible **activar y/o desactivar la confidencialidad de los datos**. La configuración de dicho proceso se define en el fichero *hdivConfig.xml* de la siguiente manera:

a. Activación de la Confidencialidad (configuración por defecto en *struts-examples*):

```
<bean id="confidentiality" class="java.lang.String">
  <constructor-arg>
    <value>true</value>
  </constructor-arg>
</bean>
```

b. Desactivación de la Confidencialidad:

```
<bean id="confidentiality" class="java.lang.String">
  <constructor-arg>
    <value>false</value>
  </constructor-arg>
</bean>
```

8.1.3 Tamaño máximo del Estado de HDIV

Como hemos visto en el capítulo 7.2.2 – *applicationContext* – es posible definir el tamaño máximo del estado a almacenar en la parte cliente tanto en la estrategia de cifrado como en la de hash, almacenando dicho estado en la parte del servidor en los casos en los que supere dicho tamaño máximo definido por el usuario.

Un valor apropiado en *struts-examples* con el que podremos ver este tipo de comportamiento (almacenamiento del estado en la parte cliente y/o servidor) es 1000. Por tanto, la configuración sería la siguiente:

- a. Estrategia de Cifrado:

```
<!--CIPHER STRATEGY -->
<beanid="dataComposerCipher"
  class="org.hdiv.dataComposer.DataComposerCipher"single
  ton="false"init-method="init">

  <propertyname="application">
    <refbean="application"/>
  </property>
  <propertyname="page">
    <refbean="page"/>
  </property>
  <propertyname="encodingUtil">
    <refbean="encoding"/>
  </property>
  <propertyname="allowedLength">
    <value>1000</value>
  </property>
  <propertyname="confidentiality">
    <refbean="confidentiality"/>
  </property>
</bean>
```

b. Estrategia Hash:

```
<!--CIPHER STRATEGY -->
<beanid="dataComposerHash"
  class="org.hdiv.dataComposer.DataComposerCipher"single
  ton="false"init-method="init">

  <propertyname="application">
    <refbean="application"/>
  </property>
  <propertyname="page">
    <refbean="page"/>
  </property>
  <propertyname="encodingUtil">
    <refbean="encoding"/>
  </property>
  <propertyname="allowedLength">
    <value>1000</value>
  </property>
  <propertyname="confidentiality">
    <refbean="confidentiality"/>
  </property>
</bean>
```

8.1.4 Peticiones de tipo Multipart

8.1.4.1 Parámetros configuración

Debido a que las aplicaciones *struts-examples* para las versiones 1.2.4, 1.2.7 y 1.2.9 de Struts (*struts-examples-1.2.4-1.0*, *struts-examples-1.2.7-1.0* y *struts-examples-1.2.9-1.0*) disponen de un formulario donde se realiza la carga (upload) de un fichero, se deben configurar los valores del bean *multipartConfig* (ver capítulo 7.2.2). Por defecto los valores configurados son los siguientes:

```
<bean id="multipartConfig"
  class="org.hdiv.config.MultipartConfig">

  <property name="maxFileSize">
    <value>250M</value>
  </property>
  <property name="memFileSize">
    <value>256K</value>
  </property>
  <property name="tempDir">
    <value>c:\tmp</value>
  </property>
</bean>
```

8.1.4.2 Gestor de peticiones multipart

Debemos configurar el controlador de Struts para que haga uso del gestor de HDIV en peticiones de tipo *multipart*. Para ello, debemos añadir la propiedad *multipartClass* al controlador de Struts en el fichero de configuración *struts-config.xml* con el siguiente valor:

```
<controller multipartClass="org.hdiv.upload.HDIVMultipartRequestHandler" />
```

Con esta configuración conseguiremos que las peticiones de tipo multipart hagan uso de HDIV tomando como parámetros de configuración los definidos en el *capítulo 8.1.4.1*.

En caso de que nuestra aplicación web disponga de varios módulos, deberemos modificar el fichero de configuración *struts-config.xml* perteneciente al módulo donde se encuentra el upload de los ficheros.

8.1.5 Loguer

Como hemos visto en el capítulo 6 – *Loguer* – es posible loguear los ataques detectados por HDIV en un fichero de log. Para ello sólo hay que añadir la librería de Log4j [9] y el fichero de propiedades *log4j.properties* al classpath de la aplicación. Por defecto estos dos ficheros se distribuyen en *struts-examples*. La librería de Log4j (*log4j-1.2.9.jar*) se distribuye en el directorio */WEB-INF/lib/* y el fichero de propiedades *log4j.properties* en */WEB-INF/classes/*.

El usuario deberá configurar la propiedad *log4j.appender.R.File* del fichero *log4j.properties* para indicar el fichero donde loguear todos los ataques detectados por HDIV. Por defecto el valor de dicha propiedad viene definida con: *C://hdiv-attacks.log*.

```
log4j.appender.R.File=C://hdiv.log
```

9. CONCLUSIONES

La mayoría de vulnerabilidades a nivel aplicación pueden ser eliminadas con una programación adecuada de las aplicaciones.

Existen problemas que pueden ser solucionados mediante librerías o funcionalidades ya existentes mientras que otras todavía requieren de soluciones propietarias para cada caso. HDIV viene a cubrir el vacío existente cubriendo gran parte de las vulnerabilidades que no disponen de soluciones estándar o uniformes que garanticen la seguridad de las aplicaciones web.

En resumen, las **ventajas ofrecidas por HDIV** respecto a las soluciones existentes en la actualidad son las siguientes:

- ✓ Soluciona de forma **transparente al programador** la verificación de la **integridad de los datos** enviados por el cliente, eliminando en consecuencia las vulnerabilidades derivadas de la alteración de los datos (parameter tampering).
- ✓ HDIV **garantiza la confidencialidad** de todos los datos no editables por el cliente (valores de los parámetros no editables, direcciones o páginas de destino). Esta propiedad evita proporcionar información clave (identificadores de BD, ..) para realizar diferentes tipos de ataques.
- ✓ Se aplica a través de configuración (web.xml) **sin modificar el código fuente** de las aplicaciones.
- ✓ Genera **logs con ataques reales**, aportando información vital para conocer el riesgo existente en las diferentes aplicaciones, incluyendo el identificador de usuario del atacante además de su IP.

10. REFERENCIAS

- [1]. Gartner, Nov 2005
<http://gartner.com>
- [2]. Studies from numerous penetration tests by Imperva
http://www.imperva.com/application_defense_center/papers/how_safe_is_it.html
- [3]. 9 Ways to Hack a Web App
<http://developers.sun.com/learning/javaoneonline/2005/webtier/TS-5935.pdf>
- [4]. Ejemplos de herramientas básicas de auditoría web:
- For Firefox (Tamper Data): <https://addons.mozilla.org/firefox/966/>
 - For Explorer: (TamperIE): <http://www.bayden.com/Other/>
- [5]. Struts
<http://struts.apache.org/>
- [6]. Spring Framework
<http://www.springframework.org>
- [7]. Open Web Application Security Project
<http://www.owasp.org/>
- [8]. Commons Logging
<http://jakarta.apache.org/commons/logging/>
- [9]. Logging Services: Log4j
<http://logging.apache.org/log4j/docs/index.html>