Seguridad en Sistemas Informáticos (SSI)

Programación Segura

Carlos Pérez Conde

Departament d'Informàtica

Escola Técnica Superior d'Enginyeria

Universitat de València

Bibliografía específica

- **OWASP Top 10 2007!**
  Project Lead: Andrew van der Stock
  Co-authors: Jeff Williams, Dave Wichers
  http://www.owasp.org/index.php/OWASP_Top_Ten_Project
- **WASC: Threat Classification**
  Co-authors: Jeff Williams, Dave Wichers
  http://www.webappsec.org/projects/threat/
- **Damn Vulnerable Linux (DVL)**
  Project Lead: Andrew van der Stock
  Co-authors: Jeff Williams, Dave Wichers
  http://www.damnvulnerablelinux.org/

OWASP Top 10 2007!

A1 – Cross Site Scripting (XSS)

A2 – Injection Flaws

A3 – Malicious File Execution

A4 – Insecure Direct Object Reference

A5 – Cross Site Request Forgery (CSRF)

A6 – Information Leakage and Improper Error Handling

A7 – Broken Authentication and Session Management

A8 – Insecure Cryptographic Storage

A9 – Insecure Communications

A10 – Failure to Restrict URL Access

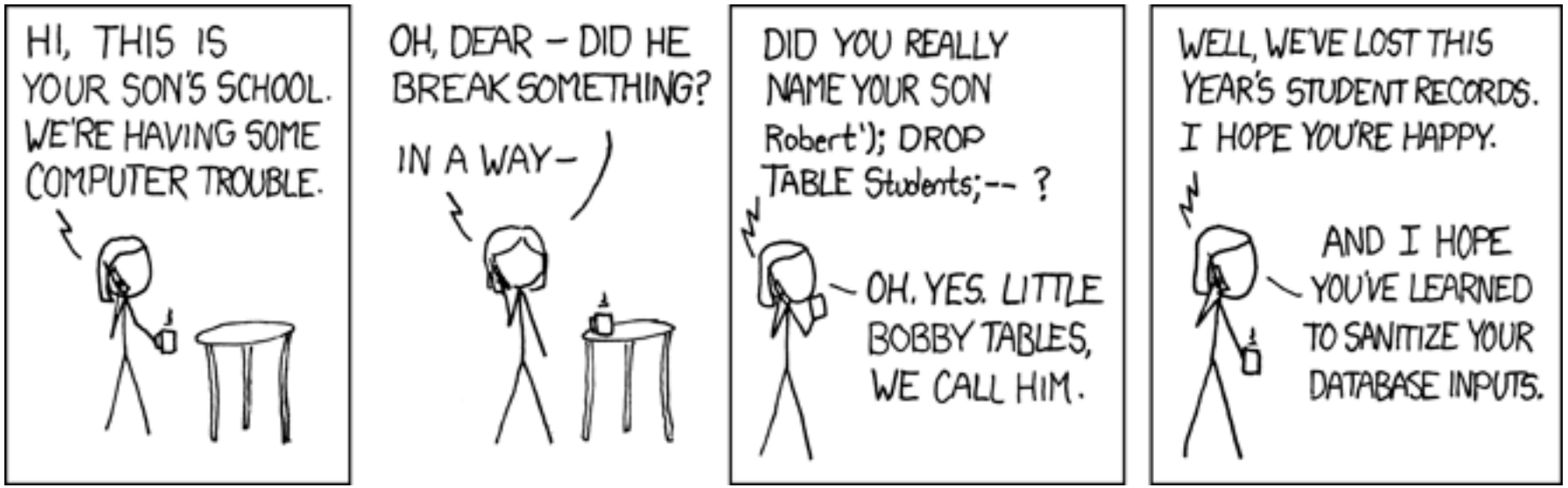
A1 – Cross Site Scripting (XSS)

- **HTML generado por el cliente es ejecutado por el navegador web**
  - reflejado: enlaces en correos, páginas web...
  - almacenado: correo web, foros, blogs...
  - inyección a través de DOM: manipulando document.URL, document.location...)
- **Solución**
  - validar y/o codificar todos los parámetros **antes** de incluirlos en páginas HTML
  - validar usando principalmente “listas blancas”
  - evitar errores de canonicalización

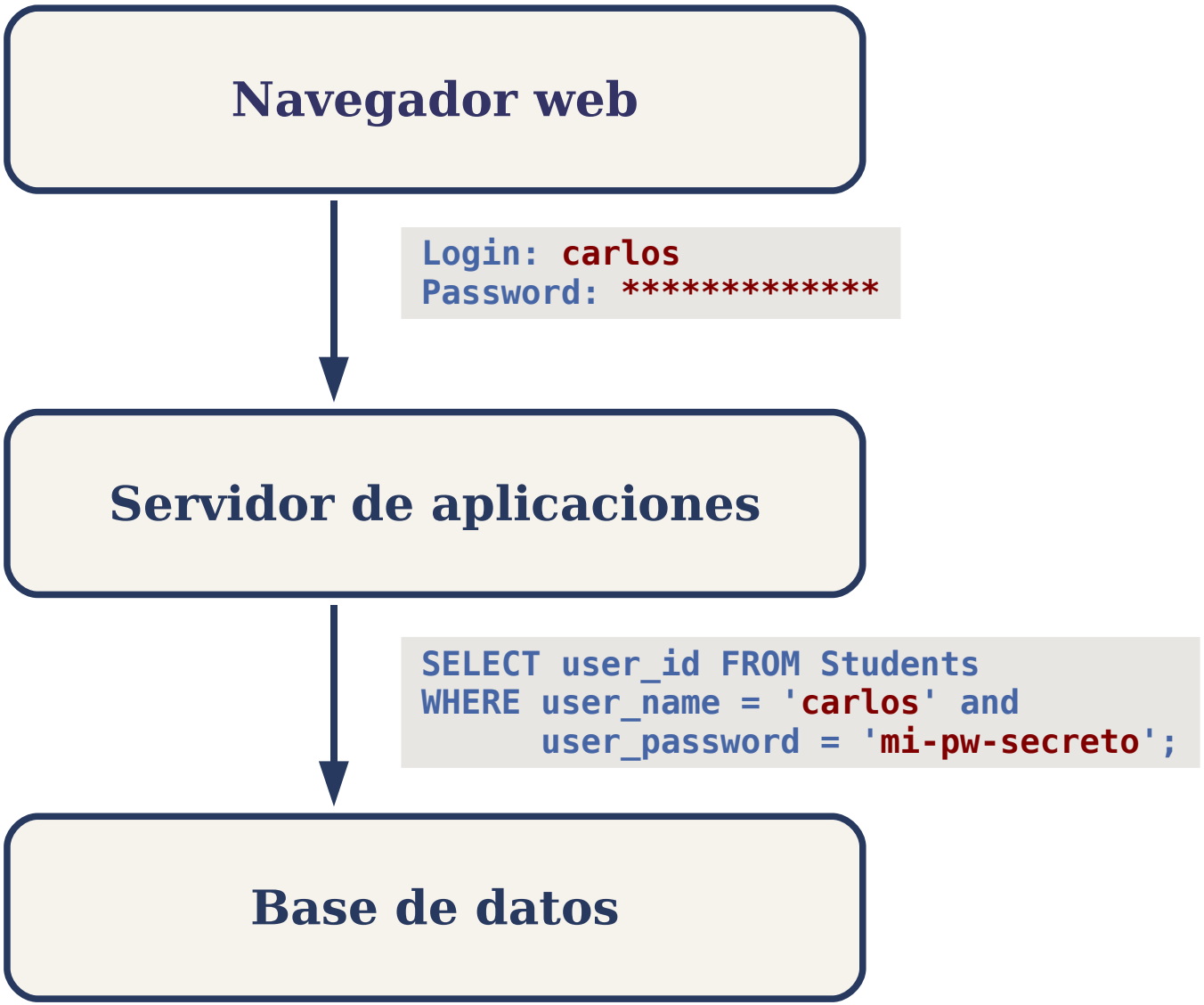
A2 – Injection Flaws (particularly SQL injection)

Exploits of a Mum

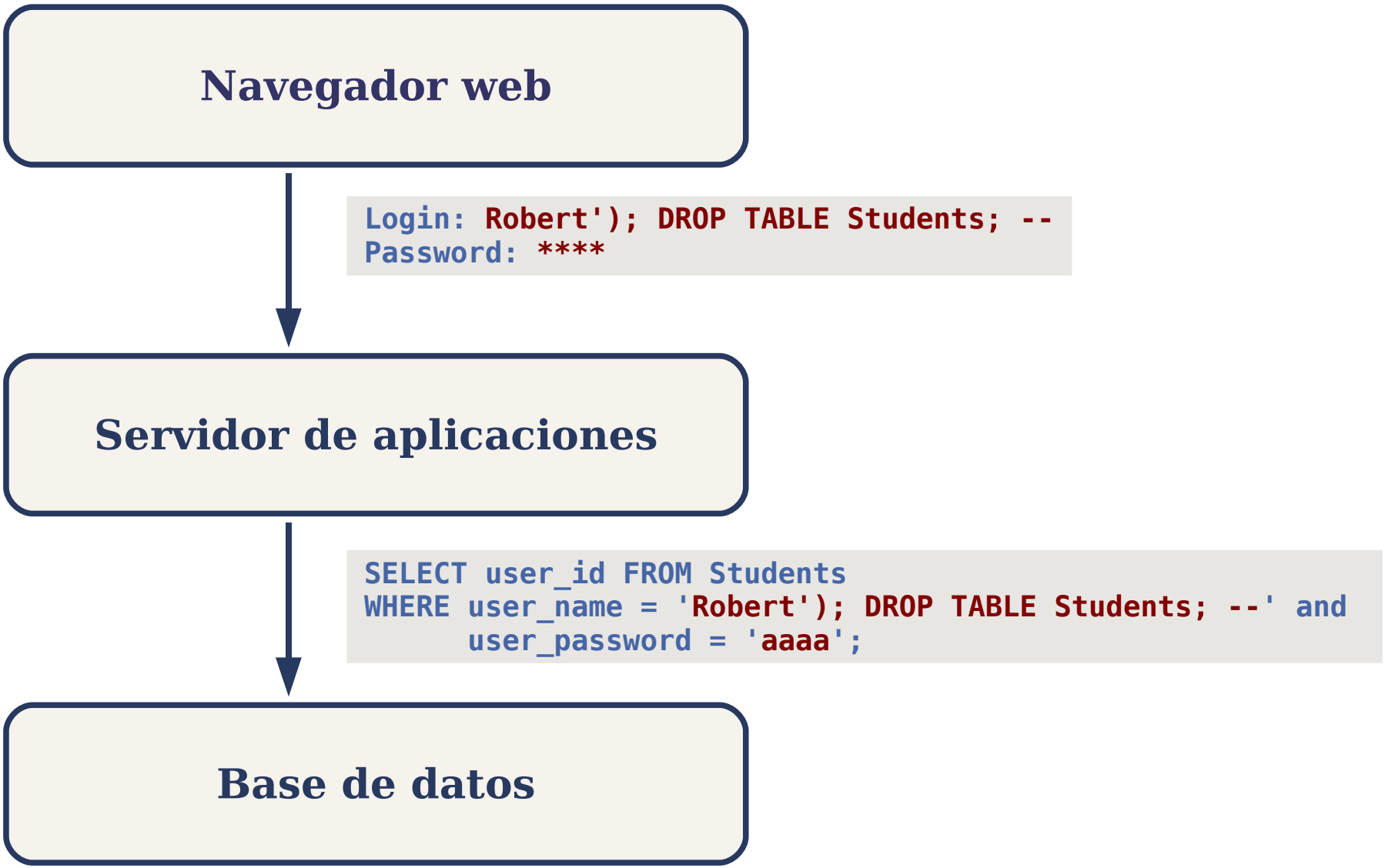
http://xkcd.com/327/



Inyección SQL



Inyección SQL



Soluciones para la inyección SQL

- No confiar en la validación realizada por el cliente
- Normalizar los valores de entrada
- Aplicar validación en el servidor
- Restringir los tipos de datos aceptados
- Codificar y validar la salida que se genera
- Usar preferentemente “listas blancas”
- Tratar de forma segura los errores
- Aplicar el principio del menor privilegio

A3 – Malicious File Execution

- **Empleo de ficheros o nombres de ficheros proporcionados por el usuario**
  - **`include $_REQUEST['filename'];`**
  - datos en ficheros de sesión, registros, imágenes subidas...
  - usar flujos como **`zlib://`** ó **`ogg://`** que ignoran **`allow_url_fopen`** ó **`allow_url_include`**
- **Soluciones**
  - usar referencias indirectas
  - diferenciar datos validados de los del usuario
  - validar la entrada usando “listas blancas”
  - filtrar los intentos de acceso remoto desde el servidor web
  - usar mecanismos de aislamiento: chroot, jail, máquinas virtuales...
  - usar mecanismos del lenguaje: tainting, allow_url_fopen...

A4 – Insecure Direct Object Reference

- **La aplicación expone una referencia a un objeto interno**
  - fichero, directorio
  - registro de una base de datos
- **Ejemplos:**

```
<select name="language"><option value="fr">Français</option></select>
…
require_once ($_REQUEST['language'].."lang.php");
```

  - manipulable con inyección del byte nulo: `"../../../../etc/passwd%00"`

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
String query = "SELECT * FROM table WHERE cartID=" + cartID;
```

  - se puede cambiar “cartID” para acceder a cualquier carro de compra

Soluciones para Insecure Direct Object Reference

- Evitar exponer las referencias
- Validar todas las referencias a objetos
- Verificar la autorización en todos los accesos

- Usar índices o mapas de referencias

```
http://www.example.com/application?file=1
```

- Verificar la autorización

```
int cartID = Integer.parseInt( request.getParameter( "cartID" ) );
User user = (User)request.getSession().getAttribute( "user" );
String query = "SELECT * FROM table WHERE cartID=" + cartID + " AND
    userID=" + user.getID();
```

A5 – Cross Site Request Forgery (CSRF)

- **Provocar que el navegador genere peticiones HTTP ocultas a recursos restringidos**
- **Se aprovecha la autentificación implícita**
  - autentificación HTTP (ej: usuario y contraseña)
  - cookies
  - autentificación SSL del cliente
  - autentificación basada en Ips
- **Ejemplo**

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

Soluciones para CSRF

- **NO funciona**
  - Usar POST en vez de GET
  - Usar cookies secretas
- **Sí funciona**
  - Usar testigos aleatorios únicos en los formularios

A6 – Information Leakage and Improper Error Handling

- **La aplicaciones filtran información sensible**
  - sobre su configuración, diseño interno...
  - datos privados a los que tienen acceso
- **La información puede ser usada para otros ataques**
- **Ejemplos:**
  - comentarios en el código fuente

```
<TR>
   <!--If the image files are missing, restart VADER -->
   <TD bgColor="#ffffff" colSpan="5" height="17"width="587"> </TD>
</TR>
```

  - mensajes de error

```
An Error Has Occurred.
Error Message: System.Data.OleDb.OleDbException: Syntax error (missing operator)
    in query expression 'username = ''' and password = 'g''. at ...
```

Soluciones para el filtrado de información

- Comprobar la aplicación con todo tipo de datos de entrada inválidos y analizar los mensajes generados
- Estrategia común para gestionar excepciones
- Deshabilitar o limitar los detalles mostrados sobre errores (especialmente de capas internas: BD, SO...)
- No usar los gestores de error por defecto
- Garantizar que los caminos de ejecución sensibles devuelven mensajes de error idénticos en más o menos el mismo tiempo (o hacerlo aleatorio)

A7 – Broken Authentication and Session Management

- **Fallo al proteger credenciales y tokens de sesión**
- **Soluciones**
  - usar SSL exclusivamente para todo acceso autenticado (A9)
  - encriptar todas las credenciales y tokens para almacenarlos (A8)
  - planificación cuidadosa
    - no exponer datos sensibles en URLs o registros
    - utilizar un único mecanismo de autentificación
    - no usar direcciones IP, consultas al DNS o “referrer headers” para autenticación
    - ser cuidadoso con el envío de contraseñas a direcciones de correo
    - limitar o eliminar el uso de cookies para la autentificación o gestión de sesiones (ej: recordar al usuario en el sitio web)
    - no aceptar id. de sesión nuevos, preestablecidos o inválidos en URLs o peticiones (evitar “session fixation attacks”)
    - crear una nueva sesión tras la autentificación o cambio de nivel de privilegio
    - proporcionar enlaces para desconectarse
    - utilizar mecanismos de autodesconexión

A8 – Insecure Cryptographic Storage

- **Fallos al encriptar datos sensibles**
  - no encriptarlos
  - utilizar algoritmos criptográficos propios
  - usar incorrectamente algoritmos fuertes
  - continuar usando algoritmos débiles (MD5, SHA-1, RC3, RC4...)
  - usar claves preprogramadas o almacenarlas desprotegidas
- **Asegurarse de cumplir la normativa vigente**

Soluciones para A8

- **Sólo almacenar lo imprescindible**
- **Usar algoritmos probados (no crear nuevos)**
  - AES, RSA para criptografía asimétrica
  - SHA-256 o mejores para “hashing”
  - http://www.owasp.org/index.php/Guide_to_Cryptography
- **No usar algoritmos débiles (ej: MD5, SHA-1)**
- **Gestión cuidadosa de claves**
  - generarlas fuera de línea
  - almacenar las claves privadas con extremo cuidado
  - nunca transmitirlas por canales inseguros
- **Cuidar todos los caminos de acceso**
  - web, ficheros, bases de datos, servidores de aplicaciones

A9 – Insecure Communications

- **Fallar al encriptar comunicaciones sensibles**
- **Asegurarse de cumplir la normativa vigente**
- **Soluciones**
  - usar SSL para conexiones autentificadas o que transmiten información sensible (credenciales, números de tarjeta de crédito, datos de salud...)
  - encriptar la comunicación en la infraestructura (con servidores de aplicaciones, bases de datos, LDAP...)
  - no permitir que se pueda pasar a un modo inseguro (ej: cuando ocurre algún fallo en las comunicaciones o falla algún componente)

A10 – Failure to Restrict URL Access

- **No permitir acceso a funciones basándose en la URL**
  - es un ejemplo de seguridad mediante oscuridad
  - URLs secretas, difíciles de adivinar...
  - evaluar el control de acceso en el cliente
- **Soluciones**
  - usar una matriz de control de acceso
  - restringir el acceso a URLs y funciones en cada paso
  - realizar pruebas de penetración
  - cuidado con ficheros incluibles y librerías
  - no asumir que los usuarios desconocen ciertas URLs o APIs
  - permitir acceso sólo a ciertos tipos de ficheros (ej: HTML, PDF...)
  - mantener actualizados los componentes que manejan datos proporcionados por usuarios (imágenes, XML, textos...)