

SEGURIDAD EN APLICACIONES

Seguridad en aplicaciones

Seguridad Informática

- Definición

*Protección provista a un sistema de información para alcanzar los objetivos de preservar la **integridad, disponibilidad y confidencialidad** de los recursos del sistema de información, incluyendo software, hardware, firmware, datos/información y telecomunicaciones*

La seguridad desde el diseño

- Diseñar el sistema con la seguridad en mente.
 - No puede ser una idea posterior (ah! y ahora vamos a controlar...)
 - Difícil “agregar” seguridad a posteriori.
- Definir objetivos de seguridad concretos y medibles.
 - Sólo algunos usuarios pueden ejecutar X. Registrar la acción.
 - La salida de la función Y debe ser encriptada.
 - La función Z debe estar disponible el 99% del tiempo.

Objetivos

- Comprender las bases funcionales de las aplicaciones web
- Comprender los problemas de seguridad de las aplicaciones web
- Diseñar e implementar mecanismos de autenticación/autorización básicos
- Comprender qué es un certificado digital y ser capaz de utilizarlo
- Detectar y solucionar problemas de seguridad simples en un aplicación web
- Referencia: "Web Application Vulnerabilities: Detect, Exploit, Prevent"

¿Qué se entiende por seguridad en las aplicaciones web?

¿Cuál es la diferencia con seguridad en la programación web?

Recuerda

- Todas las aplicaciones tienen bugs
- Si tienes bugs, tienes vulnerabilidades
- Vulnerabilidad web
 - ¿qué nos ocurre?
 - Pérdidas económicas
 - Pérdidas reputación

Arquitectura básica de una aplicación web

- Una arquitectura básica consta de cuatro componentes:
 - Navegador web
 - Servidor web
 - Aplicación
 - Almacén de datos
- Implica controlar la seguridad a distintos niveles:
 - En el cliente
 - En el servidor
 - En la aplicación
 - En las comunicaciones

Control de acceso

- Un aspecto fundamental de una aplicación web es el control de acceso de los usuarios a zonas restringidas (**funcionalidad**) de la aplicación
- Aquí intervienen dos conceptos:
 - Autenticación: Verificar que el usuario es quién dice ser
 - Autorización: Verificar que el usuario tiene permiso para acceder al recurso que está solicitando

Autenticación integrada en la aplicación

- El esquema básico para realizar el control de acceso a la aplicación también se base en pares usuario/contraseña
- La diferencia con la autenticación HTTP básica es que el formulario de solicitud de credenciales aparece integrado en la propia aplicación
- Es más flexible y permite la encriptación de la información

Persistencia de credenciales de usuario

- Una vez que el usuario se ha autenticado de forma correcta lo habitual es guardar sus credenciales en un contexto persistente (típicamente la sesión)
- Single Sign-On (SSO)
- Buenas pautas para la gestión de sesiones son:
 - Establecer un tiempo máximo de vida (session-config/session-timeout)
 - Regenerar el id de sesión cada cierto tiempo
 - Detectar intentos de ataque de fuerza bruta con identificadores de sesión
 - Requerir una nueva autenticación del usuario cuando vaya a realizar una operación importante
 - Proteger los identificadores de sesión durante su transmisión

OWASP

- OWASP (Open Web Application Security Project)
- Genera bajo licencia Open Source “guías de buenas prácticas” y sugerencias para mejorar la seguridad de las aplicaciones web desde la perspectiva de la codificación
- Ranking de las vulnerabilidades más críticas para las aplicaciones web:
 - https://www.owasp.org/index.php/Top_10-2017_Top_10

Top 10 de vulnerabilidades

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

2017

A01:2017-Injection

A02:2017-Broken Authentication

A03:2017-Sensitive Data Exposure

A04:2017-XML External Entities (XXE)

A05:2017-Broken Access Control

A06:2017-Security Misconfiguration

A07:2017-Cross-Site Scripting (XSS)

A08:2017-Insecure Deserialization

A09:2017-Using Components With Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control

A01:2021-Cryptographic Failures

A03:2021-Injection

A04:2021-Insecure Design

A05:2021-Security Misconfiguration

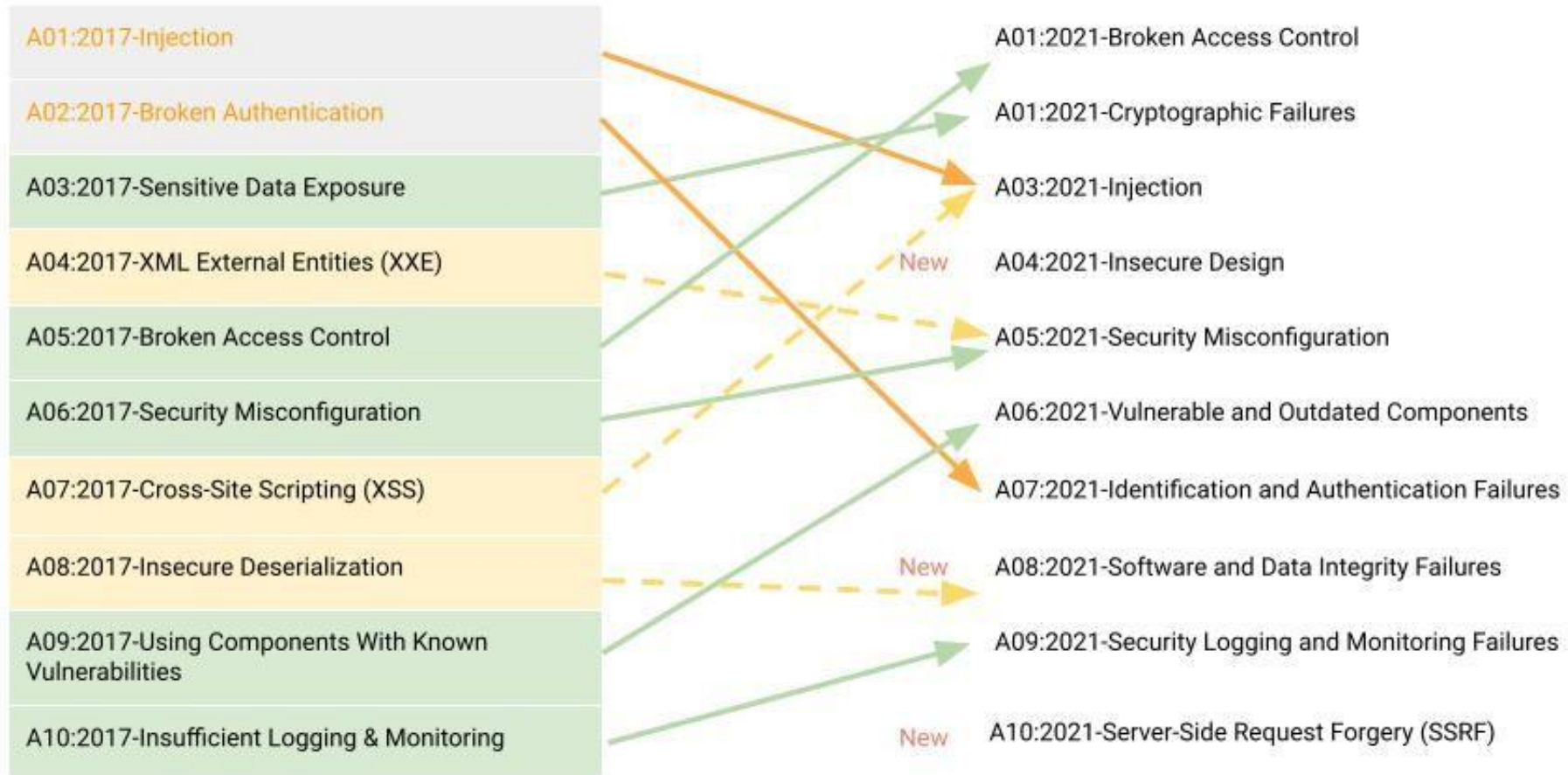
A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

A08:2021-Software and Data Integrity Failures

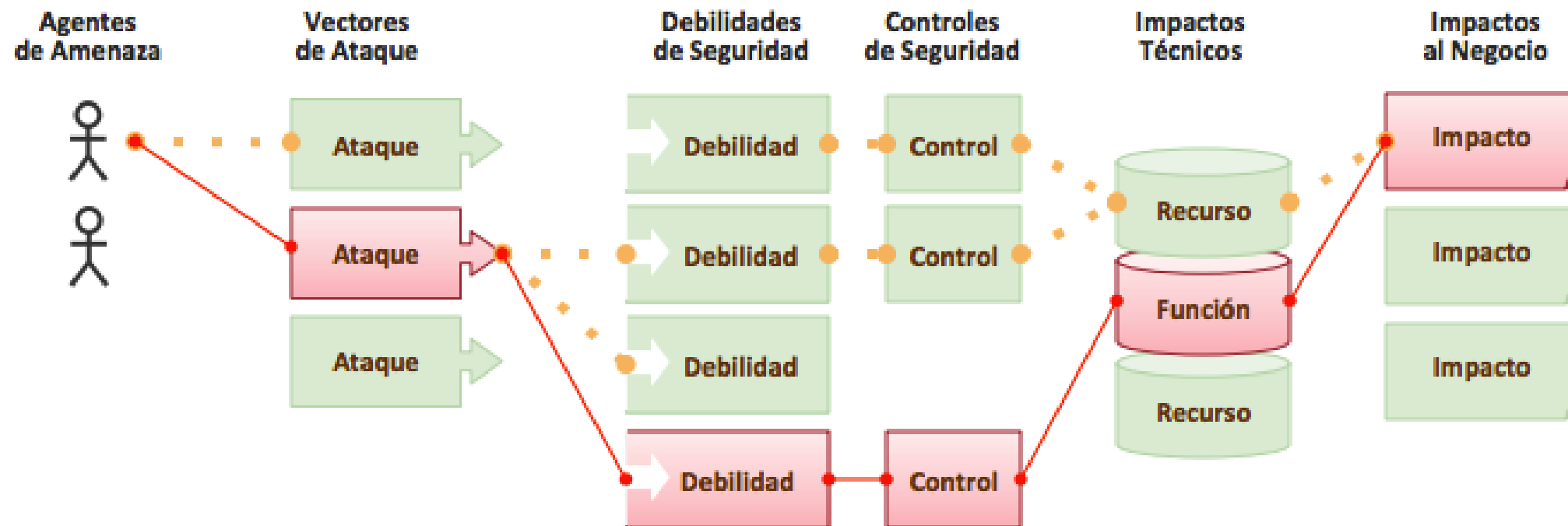
A09:2021-Security Logging and Monitoring Failures

A10:2021-Server-Side Request Forgery (SSRF)



¿Qué son los riesgos de seguridad en aplicaciones?

Los atacantes pueden potencialmente usar rutas diferentes a través de la aplicación para hacer daño a su negocio u organización. Cada una de estas rutas representa un riesgo que puede, o no, ser lo suficientemente grave como para justificar la atención.



Cross SITE Scripting (XSS)

- Código interpretado a través de entrada de usuario en un formulario o similar
- Permite el robo de sesiones, redirección, ejecución de código malicioso, ...

EJEMPLO

Código fuente:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

HTML resultante:

```
<input name='creditcard' type='TEXT' value='121241242421434'>
```

Ataque:

```
http://www.example.com/buyItem.php?CC="'><script>alert("XSS")  
;</script>'"
```


¿Cómo evitarlo?

- Validar todas las entradas de los usuarios
- XSS Prevention Cheat Sheet
 - VER: [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet)

Enumeración de nombres de usuario

- Tipo de ataque en donde el script de validación del back-end le dice al atacante si el nombre de usuario provisto es correcto (existe) o no.
 - Permite experimentar con distintos nombres.
- Dada la típica ventana de login (usuario y clave), el mensaje debe ser del estilo:

El usuario o clave son incorrectos

- Y no:

El usuario "admin" no existe

La clave no corresponde con el usuario

Inyección de SQL

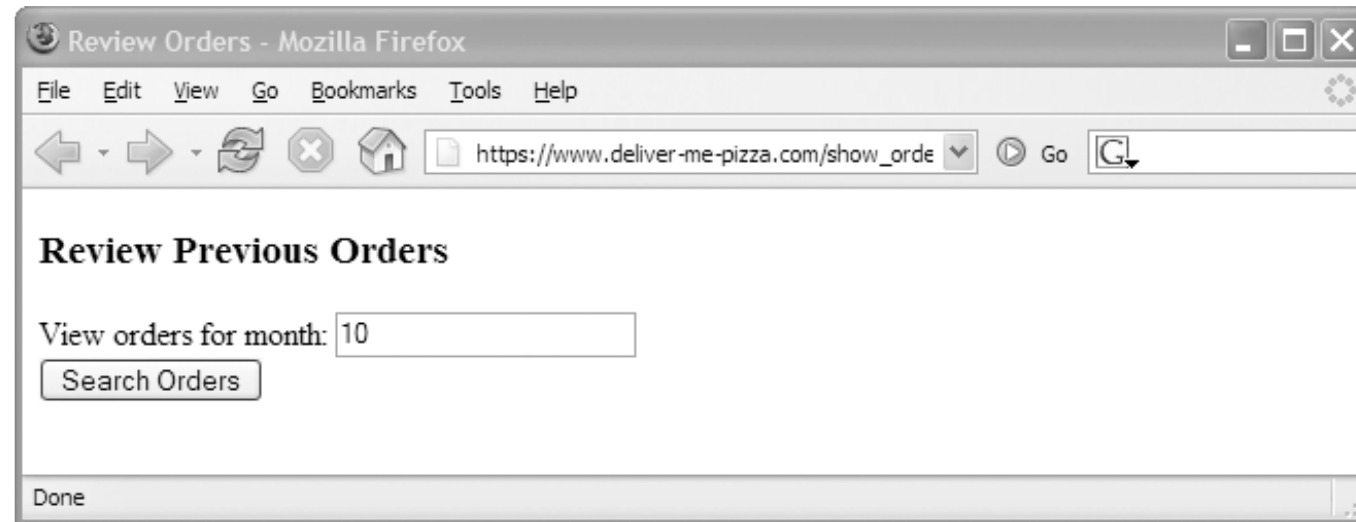
- Una aproximación vieja pero todavía popular entre los atacantes.
- Permite recuperar información vital de la base de datos de un servidor web.
- Ejemplo del impacto en casos reales:
 - El sistema de pagos de Mastercard fue atacado en junio de 2005.
 - Robaron datos de 263000 tarjetas de crédito
 - Había almacenados datos de > 40millones, sin encriptar.
- Hay varias formas de ejecutar ataque, aunque en esencia el patrón es simple.
- No todos los servidores de BB.DD. son igualmente vulnerables
 - Por ejemplo, MS SQL Server da más información de lo conveniente en los mensajes de error.

Inyección SQL

- Inserción desde parámetros de sentencias desde la URL en la aplicación ...
- <http://xxx.xxx.com/demo?id=123> or 1=1
- <http://xxx.xxx.com/demo?id=123>; DROP TABLE items;--

Escenario de ataque

- Formulario para revisar órdenes de compra de pizzas.
 - El formulario pide el (número de) mes del que se desea ver los pedidos



- Petición HTTP:

`https://www.deliver-me-pizza.com/show_orders?month=10`

Escenario de ataque (ii)

- La aplicación construye la consulta SQL a partir del parámetro

```
sql_query = "SELECT pizza, toppings, quantity, order_day " +  
            "FROM orders " +  
            "WHERE userid=" + session.getCurrentUserId() + " " +  
            "AND order_month=" + request.getParameter("month");
```

Consulta SQL Normal

```
SELECT pizza, toppings, quantity, order_day  
FROM orders  
WHERE userid=4123  
AND order_month=10
```

- Tipo de ataque 1: ingresa month='0 OR 1=1'
- Se transforma en URL: (blanco → %20, = → %3D)

```
https://www.deliver-me-pizza.com/show_orders?month=0%20OR%201%3D1
```

Escenario de ataque (iii)

Consulta maliciosa

```
SELECT pizza, toppings, quantity, order_day  
FROM orders  
WHERE userid=4123  
AND order_month=0 OR 1=1
```

- La condición WHERE es siempre verdadera!
 - El AND precede al OR
 - Ataque tipo 1: obtiene acceso a los datos privados de otros usuarios!

**All User Data
Compromised**



The screenshot shows a Mozilla Firefox browser window titled "Order History - Mozilla Firefox". The address bar is empty. The menu bar includes File, Edit, View, History, Bookmarks, ScrapBook, Tools, and Help. The main content area is titled "Your Pizza Orders:" and displays a table with the following data:

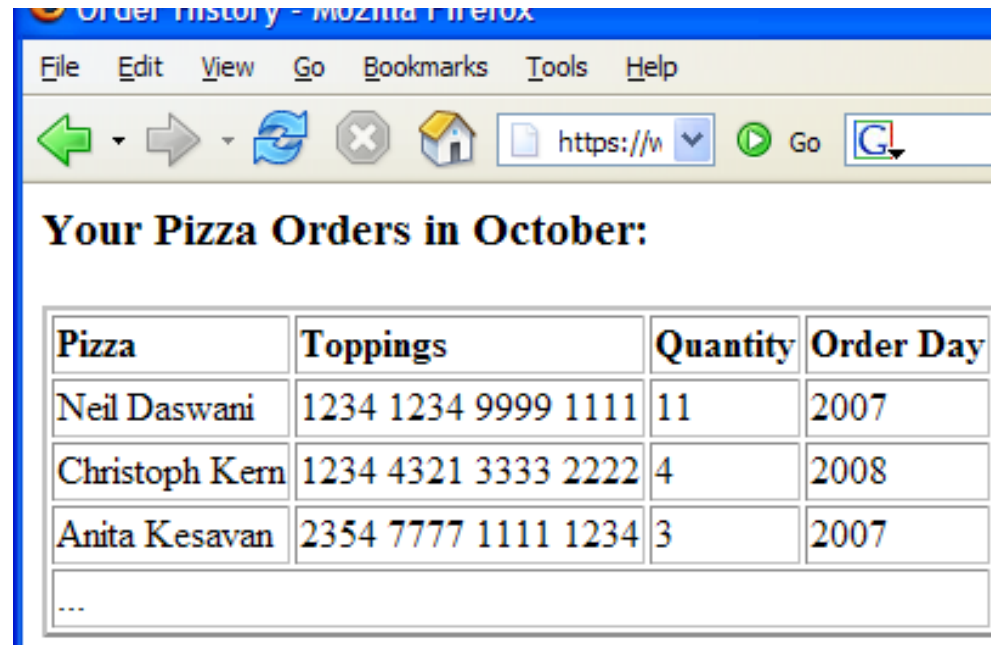
Pizza	Toppings	Quantity	Order Day
Diavola	Tomato, Mozarella, Pepperoni, ...	2	12
Napoli	Tomato, Mozarella, Anchovies, ...	1	17
Margherita	Tomato, Mozarella, Chicken, ...	3	5
Marinara	Oregano, Anchovies, Garlic, ...	1	24
Capricciosa	Mushrooms, Artichokes, Olives, ...	2	15
Veronese	Mushrooms, Prosciutto, Peas, ...	1	21
Godfather	Corleone Chicken, Mozarella, ...	5	13
...			

Escenario de ataque (iv)

- Ataque puede causar mayor daño:

```
month = 0 AND 1=0
UNION SELECT cardholder, number, exp_month, exp_year
FROM creditcards
```

- El atacante puede combinar dos consultas.
 - 1ª consulta: tabla vacía (falla condic.)
 - 2ª consulta: número de tarjeta de todos los usuarios!!



The screenshot shows a web browser window titled "Order history - Mozilla Firefox". The address bar displays "https://w". The page content shows the heading "Your Pizza Orders in October:" followed by a table with four columns: "Pizza", "Toppings", "Quantity", and "Order Day". The table contains three rows of data and a final row with three dots "...".

Pizza	Toppings	Quantity	Order Day
Neil Daswani	1234 1234 9999 1111	11	2007
Christoph Kern	1234 4321 3333 2222	4	2008
Anita Kesavan	2354 7777 1111 1234	3	2007
...			

Escenario de ataque (v)

- Aún peor, el atacante hace:
- Entonces la B.D. ejecuta
 - Ataque tipo 2: elimina la tabla creditcards de la base de datos!!
 - Próximos pedidos van a fallar: DoS!
- Sentencias problemáticas:
 - Las que modifican contenido:
INSERT INTO admin_users VALUES ('hacker',...)
 - Administrativas: shut down DB, control OS...

```
month=0;  
DROP TABLE creditcards;
```

```
SELECT pizza, toppings,  
quantity, order_day  
FROM orders  
WHERE userid=4123  
AND order_month=0;  
DROP TABLE creditcards;
```

Escenario de ataque (vi)

- Inyectar parámetros tipo string: búsqueda de ingredientes

```
sql_query =  
    "SELECT pizza, toppings, quantity, order_day " +  
    "FROM orders " +  
    "WHERE userid=" + session.getCurrentUserId() + " " +  
    "AND topping LIKE '%" + request.getParamenter("topping") +  
    "%' ";
```

- El atacante envía: `topping=brzfg%'; DROP table creditcards; --`

- La consulta resultante:
 - SELECT: tabla vacía
 - -- transforma en comentario el final
 - La info sobre tarjetas de crédito es eliminada

```
SELECT pizza, toppings,  
quantity, order_day  
FROM orders  
WHERE userid=4123  
AND topping LIKE  
'%brzfg%';  
DROP table creditcards; --  
%'
```

riesgos

- Saltar autenticación
- Volcar nuestra bbdd
- Borrar o modificar bbdd
- Ejecutar comandos de sistema

¿Cómo evitarlo?

- Revisar código y sanearlo...
 - Validación con expresiones regulares
 - Especificar la codificación de caracteres
 - La extensión de filtrado de PHP
- Ejemplo: <https://diego.com.es/filtrado-de-datos-de-entrada-en-php>
- https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Contramedidas

- Evitar conectarse a la BD como root o como propietario
 - Usar usuarios específicos con permisos restringidos.
- Validación de entradas y escape de caracteres especiales.
 - Mejor métodos *whitelist* que *blacklist*
- Uso de *prepared statements*
- Mitigar impacto

Desventajas del método

- Filtrar comillas, puntos y comas, espacios en blanco, y... ?
 - Nos podemos olvidar de un char peligroso
 - La listas negras no son una solución “total”.
 - Por ejemplo, no se puede utilizar con parámetros numéricos.
- Puede tener conflicto con requisitos funcionales
- ¿Qué pasa si queremos guardar un usuario **O’Brien**?

Validación de entradas por listas blancas

- *Lista blanca*: sólo se permiten entradas dentro de un conjunto bien definido de valores seguros:
 - Se verifica la entrada con expresiones regulares.
- Ejemplo: el parámetro `month` debe ser un entero no negativo:
 - RegExp: `^[0-9]*$` - 0 o más dígitos
 - Los símbolos `^` y `$` hacen referencia al principio y final de la cadena.
 - `[0-9]` acepta un dígito, `*` especifica 0 o más veces

Escape de caracteres especiales

- Es posible escapar las comillas en lugar de eliminarlas.
- Por ejemplo, el usuario envía:

`user o'connor, password terminator`

```
sql = "INSERT INTO USERS (uname,passwd) " +  
      "VALUES (" + escape(uname) + "," +  
      escape(password) + ")" + ";
```

- `escape(o'connor) = o''connor`

```
INSERT INTO USERS (uname,passwd)  
VALUES ('o''connor','terminator');
```

- Sólo funciona para entradas en formato cadena
- Parámetros numéricos siguen siendo vulnerables.

Mitigando el efecto de los ataques

- Prevenir el filtrado de meta información
- Limitar los privilegios (defensa en profundidad).
- Encriptar datos sensibles almacenados en la B.D.
- Endurecer las defensas del servidor de B.D. y del S.O. anfitrión.
- Validar las entradas

Prevenir el filtrado de meta información

- Conocer el esquema de la B.D. facilita el trabajo del atacante.
- *Blind SQL Injection*: el atacante intenta interrogar al sistema para recrear su organización.
- Prevenir filtrado de información de estructura:
 - No muestre mensajes de error detallados y secuencia de llamadas (*stack traces*) a usuarios externos.
- `Select * from tabs;`
- `Desc tabla;`

Limitación de privilegios

- Aplicar el principio de privilegio mínimo.
 - Permisos de acceso, qué tablas/vistas el usuario puede consultar.
 - Comandos: puede ejecutar updates/inserts?
- No dar más privilegios que los que el usuario típico necesite.
- Ejemplo: puede evitar que un atacante ejecute sentencias INSERT y DROP.
 - Pero no puede evitar que haga SELECT y comprometa información sensible de usuarios.
 - No es una solución completa, pero disminuye el daño.

Encriptado de información sensible

- Encriptar la información almacenada en la B.D.
 - Segunda línea de defensa.
 - Sin la clave, el atacante no puede leer información sensible.
- Precauciones en las gestiones de claves: no almacenar las claves en la BD!!
- Algunas BD permiten un encriptado automático de los datos, pero retornan en plano el resultado de las consultas!

Endurecer las defensas del servidor y S.O.

- Funciones peligrosas pueden estar disponibles por omisión.
- Por ejemplo, en MS SQL Server:
 - Se permite a los usuarios abrir sockets de entrada y salida.
 - Un atacante puede robar datos, cargar binarios, hacer scanning de puertos.
- Deshabilitar servicios no usados y cuentas en el S.O.
 - Por ejemplo, no hay necesidad de un servidor web en un anfitrión dedicado a servidor de B.D.

Validar las entradas

- La validación de los parámetros de las consultas no es suficiente.
- Validar todas las entradas apenas se conozcan en el código.
- Rechazar entradas demasiado largas
 - Puede prevenir algún error de buffer overflow en el parser de SQL
- La redundancia ayuda a proteger sistemas
 - Por ejemplo si el programador olvida validar las entradas de las consultas.
 - Dos líneas de defensa

Manipulación del estado del cliente

- Consecuencia de la característica del protocolo HTTP de ser *stateless*:
 - Para llevar a cabo transacciones, las aplicaciones web deben mantener estado.
 - Con frecuencia, el estado se envía a los clientes que los devuelven al servidor en futuras solicitudes.
- Ejemplo de explotación:
 - Los parámetros escondidos de HTML no son en realidad tan escondidos, pueden ser manipulados.
 - Cookies
 - Parámetros hidden

Usar POST en lugar de GET

- GET: los parámetros (en especial id sesión) son revelados en URL
- Referers pueden revelar información a través de enlaces:
 - Este enlace ``
 - Envía solicitud (al ser pinchado):

```
GET / HTTP/1.1 Referer:  
https://www.deliver-me-pizza.com/submit_order?  
session-id=3927a837e947df203784d309c8372b8e
```

- El id de sesión es enviado al sitio `grocery-store-site` y seguramente almacenado en logs!

Beneficios del POST

- Solicitud
POST

```
POST /submit_order HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 45

session-id%3D3D3927a837e947df203784d309c8372b8e
```

- Id de sesión no visible en URL
 - Copy-paste no revela nada
 - Pequeños inconvenientes para el usuario, pero más seguro.
- Referers pueden revelar información, incluso sin intervención del usuario
 - En vez de enlace, imagen:

 - La petición GET x banner.gif revela id de sesión!

Cookies

- *Cookie* – porción de estado mantenido por el cliente
 - El servidor envía la cookie al cliente
 - El cliente la retorna en cada petición HTTP
 - Ejemplo: id de sesión en cookie

```
HTTP/1.1 200 OK  
Set-Cookie: session-id=3927a837e947df203784d309c8372b8e; secure
```

- Por seguridad se debe usar SSL
- El navegador responde:

```
GET /submit_order?pay=yes HTTP/1.1  
Cookie: session-id=3927a837e947df203784d309c8372b8e
```

Problemas con las Cookies

- Se asocian al navegador
 - Se envían con cada petición, no hace falta incluir campo oculto.
- Si el usuario no hace log out, el atacante puede usar el mismo navegador para suplantar identidad.
- Los ids de sesión deben tener tiempo de vida limitado.

JavaScript

- Ejemplo: cálculo coste de un pedido

```
<html><head><title>Order Pizza</title></head><body>
  <form action="submit_order" method="GET" name="f">
    How many pizzas would you like to order?
    <input type="text" name="qty" value="1" onKeyUp="computePrice();">
    <input type="hidden" name="price" value="15.50"><br>
    <input type="submit" name="Order" value="Pay">
    <input type="submit" name="Cancel" value="Cancel">
    <script>
      function computePrice() {
        f.price.value = 15.50 * f.qty.value; // compute new value
        f.Order.value = "Pay " + f.price.value // update price
      }
    </script>
  </body></html>
```

JavaScript (ii)

- Usuario malicioso puede borrar código JavaScript, sustituir los parámetros y enviar

```
GET /submit_order?qty=1000&price=0&Order=Pay
```

- **Cuidado:** el servidor no debe confiar en la validación de datos o cálculos hechos en JavaScript.
 - Muy fácil alterar script en cliente.
 - Debe ser rehecho en el servidor

Resumen manipulación estado del cliente

- No confiar en las entradas del usuario!
 - Mantener el estado en el servidor (costoso en espacio y tiempo).
 - Firmar parámetros de la transacción (costoso en ancho de banda)
 - No es suficiente con validar y calcular en JavaScript

Gestión de Sesiones

- Riesgo:
 - Robo de cookie de sesión
 - Con la cookie te puedes hacer pasar por la víctima
 - ¿mi web dispone de mecanismos para evitarlo?

Gestión de Sesiones

- Caducidad
- Timeout por inactividad
- Cierre de sesión, destruir cookie
- Nuevo login = nueva cookie
- Limitar sesiones concurrentes

- Cookie en URL
- Cookie impredecible y resistente a modificaciones
- Transmitir por canal seguro HTTP Only o Secure
- Ejemplo: `http://x.x.x.x/login;jsessionid=ewqeqwewqewqeqweqw`