UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Máster Universitario en Ingeniería Informática

CyberSeguridad
Tema 4b: Fallos Web

José Ismael Ripoll Ripoll

Máster Oficial Universitario en Ingeniería Informática
muiinf.webs.upv.es

MUIInf

UII

Contenidos

- SQL injection (Ataque)
- Cross-Site-Scripting (Ataque)
- Same Origin Policy (Defensa)
- Cross-Site-Request-Forgery (Ataque)

- Las vulnerabilidades de seguridad más comunes y relevantes en aplicaciones Web en los últimos años son:
  - Inyección SQL
  - XSS (Cross-Site Scripting)
  - CSRF (Cross-Site Request Forgery)
  - Otros ataques de inyección, sobre XPath (el lenguaje de consulta de información en repositorios de XML) y LDAP (servicio de directorio) Publicación de información sensible
  - HTTP Response Splitting
  - Path traversal
  - Falta de autenticación en objetos sensibles

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
  - A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application.
- ¿Qué es SQL?
  - SQL: Structured Query Language.
  - Permite acceder y manipular las bases de datos.
  - Es un estándar ANSI, usado por la mayoría de DB.
  - En la aplicaciones web se utiliza para guardar los datos.
- ¿Cómo se accede a la base de datos desde la web?
  - No se ataca al servidor web ni al browser.
  - Se explota la "incorrecta validación los datos del usuario"

¿Para qué sirve el SQL?

- Ejecutar consultas a la base de datos.
- Obtener información de la DB.
- Insertar nuevos datos (registros) en el DB.
- Eliminar registros de la DB.
- Crear nuevas bases de datos.
- Crear tablas en una DB.
- Almacenar rutinas o procedimientos en la DB.
- Crear vistas de las DB.
- Gestionar permisos en tablas, rutinas, vistas, etc.
- etc.

SQL injection

- Las sentencias SQL son cadenas de texto que se envían al motor de base de datos para que las analice y las ejecute: **`Select`**, **`Insert`**, **`Delete`**, **`Union`**, etc...
- Las sentencias SQL contienen los comandos SQL junto con los datos que se necesitan.

```
"SELECT * FROM Users WHERE UserId = Pedro"
```

- Se suelen invocar desde un programa php u otro lenguaje de programación del servidor.

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

http://www.w3schools.com/sql/sql_injection.asp

- La inyección SQL es una técnica que permite a un atacante insertar comandos SQL dentro de una sentencia SQL a través de una página web.

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

- Si la variable **txtUserId** tuviera el valor: **105 or 1=1**

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

- Que daría como resultado: **CIERTO**.

- Además de modificar la sentencia SQL, es posible añadir nuevos comandos:

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

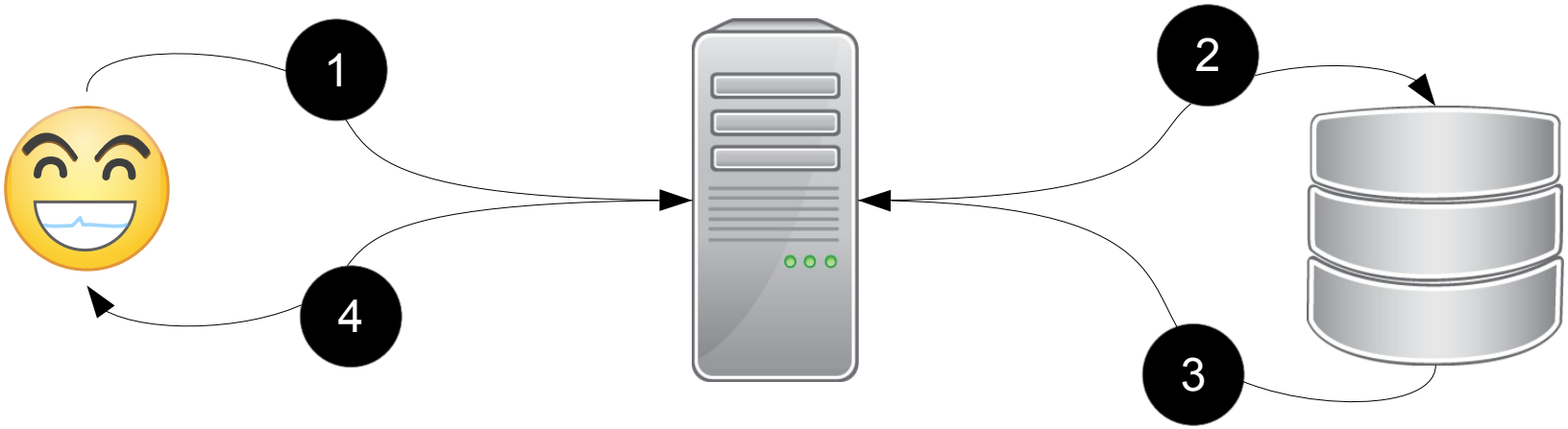
- Si **txtUserId** vale: **105; DROP TABLE Suppliers**

```
SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers
```

- Lo que borra la base de datos de Suppliers.

Envía una petición con un campo malformado

El código del servidor construye una senticna SQL incorrecta.



El funcionamiento del servidor es controlado por el atacante

La BD retorna los datos que le interesan al atacante

- Validar los datos introducidos por el usuario:
  - Limpiar la entrada del usuario de caracteres que se puedan confundir con la sintaxis SQL.
- No utilizar las sentencias SQL directamente:
  - No construir sentencias SQL mediante operaciones con cadenas, desde el servidor web.

Escribir código correcto

- ¿Por qué se produce el fallo?
  - Porque se confunden datos con código SQL.
  - El manejo de cadenas por parte del servidor WEB no es correcto.
  - Por otra parte, es un problema causado por la "excesiva" proximidad entre datos y código.
- Confusión entre datos y comandos!
  - Ejemplo de liveoverflow
  - https://www.youtube.com/watch?v=WWJTsKaJT_g
- Formas de corregirlo:
  - Validar los datos introducidos por el usuario.
  - No utilizar las sentencias SQL directamente.

Mitigación de SQL Injection

- Funciones de filtrado frente a ataques de inyección SQL:
  - El filtrado depende en gran medida del tipo de base de datos empleada.
  - Se debe filtrar el envío de caracteres especiales para la base de datos y palabras SQL reservadas, como comentarios (--, #, /* o */), ";" (fin de consulta SQL), caracteres comodín (*, %, o _), concatenación (+ o ||), u operadores SQL: OR, TRUE, 1=1, SELECT, JOIN, UPDATE, INNER, INSERT, PRINT, waitfor delay "xx", etc.
  - Adicionalmente es necesario filtrar todas las representaciones de los caracteres especiales. Por ejemplo, la "'" es igual a %27 o el "=" es %3D. Asimismo, deben traducirse las funciones propias de la base de datos, como la función CHAR(). Por ejemplo, CHAR(77) es igual al carácter 'M' (valor ASCII decimal: 77).

Mitigación de SQL Injection

- Funciones de filtrado frente a ataques de inyección SQL:
  - Criterios de filtrado similares deben aplicarse a la inyección de comandos en otros lenguajes de consulta, como por ejemplo LDAP, con sus operadores asociados: AND (&), OR (|),NOT (!), <=, >=, = y ~= y el carácter comodín (*).
  - Otro de los lenguajes de consulta comúnmente empleados en la actualidad en servicios Web es XPath, que permite la consulta de repositorios de datos en XML.
  - Una de las técnicas recomendadas para evitar la inyección SQL es la utilización de procedimientos almacenados, donde la entrada del usuario se convierte en parámetros específicos del procedimiento. Deben filtrarse los parámetros correctamente y no emplear SQL dinámico en el procedimiento almacenado para evitar ataques de inyección.

No utilizar SQL directamente

- Utilizar funciones ya construidas en el lenguaje para hacer operaciones SQL: "parametrized queries".
  - No es tan potente como el SQL nativo.
  - Es necesario escribir más código.
  - Cada lenguaje lo puede implementar de forma diferente.
- Por ejemplo PHP define PDO (PHP Data Objects)

```
 $stmt = $dbh->prepare("SELECT * FROM users WHERE USERNAME = ? AND
PASSWORD = ?");

$stmt->execute(array($username, $pass));
```


Cross Site Scripting XSS

§ CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting', XSS)

- La aplicación Web envía los **datos previamente proporcionados por otro usuario** al navegador Web sin realizar ninguna validación o codificación del contenido. Este ataque permite a un atacante ejecutar código (scripts) en el navegador de la víctima: robo de sesiones, modificación de los contenidos de la Web y su configuración.
- Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

Cross-site Scripting (XSS)

3

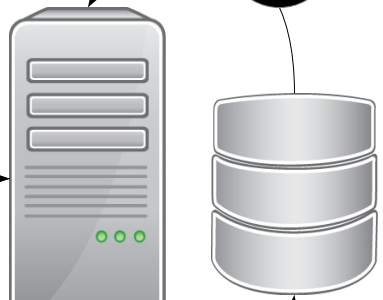
La víctima solicita una página legal, pero...

En la respuesta se añade el código malicioso

4

5

Recibe código malicioso



1

El atacante envía una petición que inyectará código javascript malicioso

2

La aplicación lo inserta en la DB

Principales XSS subtipos de ataques

- Stored XSS Attacks (persistente)
  - El atacante "añade" código de scripting en las páginas que ofrece el servidor. Típicamente mediante formularios wiki o páginas de comentarios.
  - https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_%28OWASP-DV-002%29
- Reflected XSS Attacks (No persistente)
  - La página que contesta el servidor contiene parte de la información de la URL (refleja parte de la petición). Se suele utilizar a través de correos-e.
  - https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_%28OWASP-DV-001%29

Mitigación del XSS

- Se debe filtrar cualquier contenido recibido por el usuario que tenga validez en el lenguaje HTML:
  - **<script>, <img>, <a href=...>, <object>, <iframe>**, etc.
  - Los caracteres de creación de etiquetas HTML, "<" y ">", y sus múltiples representaciones: < y > , %60 y %62;, < y >, junto a otros caracteres propios de código de scripts: = " ' ( ) ; &.
  - Aquellas etiquetas que puedan contener referencias: src="...".
  - Se recomienda filtrar no sólo la entrada del usuario, sino también la salida de la aplicación, para eliminar aquellos contenidos no deseados de la misma, como por ejemplo scripts o **iframes** qué referencian sitios Web remotos.
- https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

Same Origin Policy SOP

Same Origin Policy (SOP)

- Previene que un documento o script cargado de un "origen" (proto://host:port) pueda acceder o modificar propiedades del documento a no ser que tanto el documento como el script provengan del **mismo sitio.**
  - Se trata de uno de los **conceptos de seguridad más importantes** de los navegadores.
  - Se implementa en el navegador.
  - Impide que un atacante pueda utilizar los ordenadores de los clientes para realizar ataques.
  - Impide que javascript de terceros pueda acceder a cookies:

    "*let users visit untrusted web sites without those web sites interfering with the user's session with honest web sites.*"

https://www.w3.org/Security/wiki/Same_Origin_Policy

§ Ejemplo de protección de SOP:

- Juan entra en la página de su banco "**topami.com**".
- Juan se identifica en su banco y se conecta.
- Juan abre otra pestaña para entrar en una tienda a comprar zapatos: "**toa100.com**".
- La página de "**toa100.com**" tiene código JavaScript para intentar robar datos de todos los clientes que entran.

SOP evita esto

- Mediante AJAX (XMLHttpRequest()), hace peticiones a varios bancos (entre ellos a "**topami.com**").
- El banco responde con los datos de Juan porque la petición AJAX **incluye automáticamente las cookies/credenciales** que Juan tiene aún activas **por tener la sesión abierta** en su banco.
- El script de la página "**toa100.com**" recibe los datos del banco y se los envía a Malone para realizar sus fechorías.

Same Origin Policy

- Por ejemplo: un documento obtenido desde:
  - http://example.com/doc.html
- Intenta acceder al DOM obtenido desde:
  - https://example.com/target.html
- El navegador bloqueará el acceso porque el "origen" es diferente!
  - (**http**, example.com, 80) ≠ (**https**, example.com, 443).

Same Origin Policy

- In practice, there is no single same-origin policy, but rather, a set of mechanisms with some superficial resemblance, but quite a few important differences.
  - https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy
  - Same-origin policy for DOM access.
  - Same-origin policy for XMLHttpRequest
  - Same-origin policy for cookies
  - Same-origin policy for Flash
  - Same-origin policy for Java
  - Same-origin policy for svg
  - file:/// (mismo directorio)

Cross-Site Request Forgery CSRF

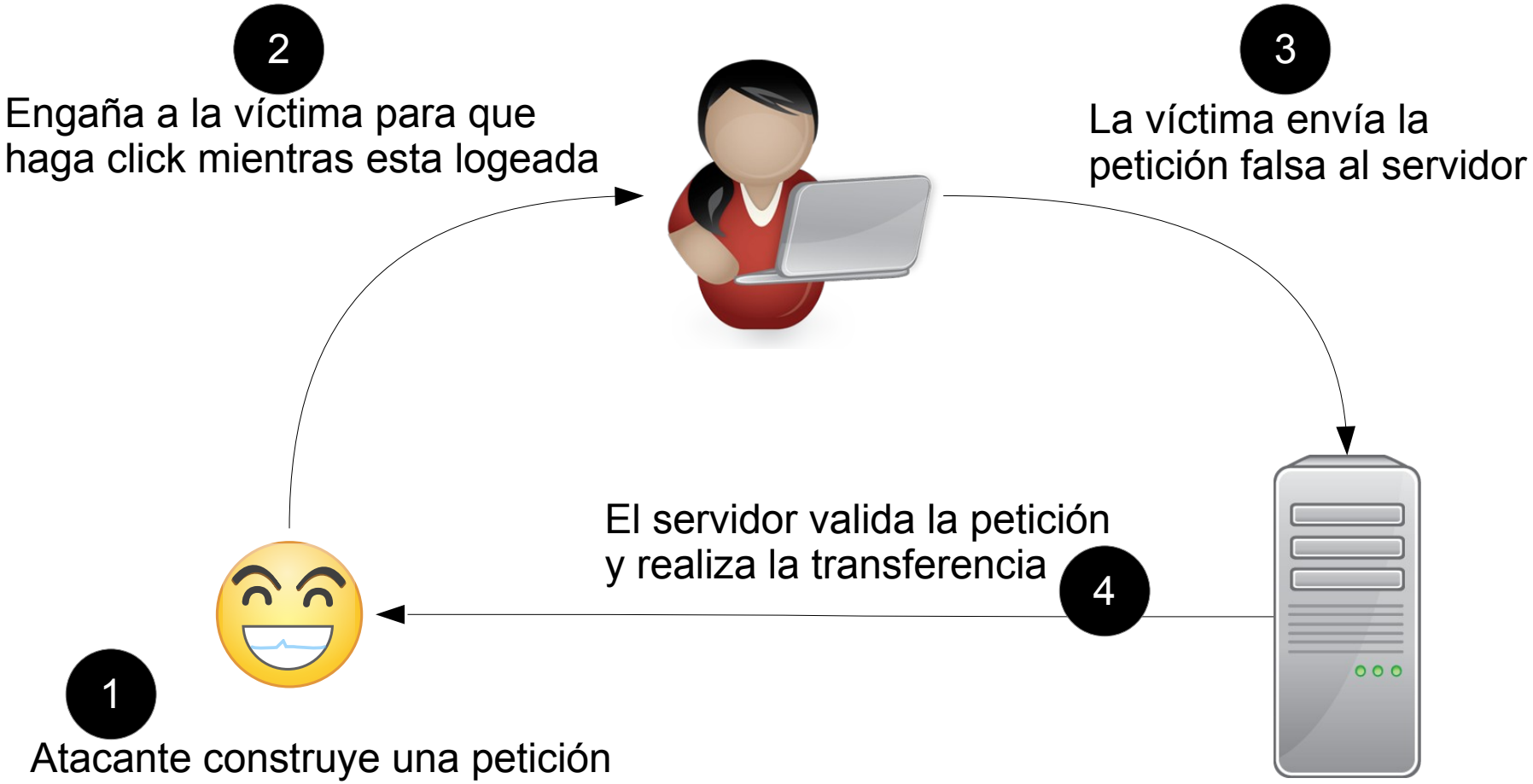
Cross-Site Request Forgery (CSRF)

- CWE-352: Cross-Site Request Forgery (CSRF)
- The web application does not, or can not, sufficiently verify whether a well-formed, valid, consistent request was **intentionally** provided by the user who submitted the request.
- Ataque que consiste en engañar a la victima (o lanzar una petición mediante javascript), **cuando está utilizando un navegador**, para que siga una URL maliciosa.
- Esta URL normalmente realizará una acción indeseada sobre un servidor en el que la víctima tenga permisos (que esté logeada).

Cross Site Request Forgery (CSRF)

- Una gestión en el banco consiste en:
  1) Logearse en el web del banco. (escribimos la url)
  2) Seleccionar las cuentas. (clicamos en los enlaces)
  3) Realizar la petición de transferencia (clicamos en los enlaces)
- Si un atacante consigue que hagamos clic en un enlace de un anuncio o un mail después de 2… premio.

Cross Site Request Forgery (CSRF)



Cross Site Request Forgery (CSRF)

- ¿Qué diferencias hay entre el CSRF del XSS?
- El XSS es una inyección de javascript que se ejecuta como si fuera de la propia página a la que accedo.
  - En este caso, las acciones realizadas por el javascript malicioso (inyectado en la página) no se puede diferenciar de las acciones del usuario.
- Por otra parte, el **CSRF es una acción que solicita una página sobre otro recurso distinto del que proviene.** Una página trata de acceder a recursos de otro site (cross-site).

Cross Site Request Forgery (CSRF)

- Tenéis que apreciar la sutiliza de este hecho:
  - ¿Cuándo es una decisión consciente del usuario?
- De hecho es un problema parecido al Control Flow Integrity. El atacante toma el control del intercambio de mensajes mientras estamos en una sesión.
- Los ataques CSRF tratan de modificar el estado de los servidores.
- El atacante no espera recibir información directa de la víctima, como exfiltración de información o un reverse shell, **porque el SOP lo bloquea**.
- https://owasp.org/www-community/attacks/csrf

CSRF, protección

- La mejor protección está en el lado del servidor. El cliente lo tiene más difícil.
- Consiste en añadir un valor aleatorio, normalmente en un campo oculto de las páginas html, de tal forma que cada página enviada al cliente esté "marcada".
- Cuando el cliente haga clic sobre cualquier enlace de esa página, el campo oculto se envía de vuelta al servidor, que deberá comprobarlo.
- Las peticiones generadas desde una URL fuera de esa sesión NO contendrán el campo oculto y el servidor lo detectará.

Lista de Comprobación

- Es conveniente tener un "checklist" que permita sistematizar todas las comprobaciones de seguridad cuando se diseña, implementa, mantiene o adquiere un servicio WEB.
- Se puede encontrar una herramienta completa en el **Anexo B de la "GUÍA DE SEGURIDAD DE LAS TIC (CCN-STIC-812) SEGURIDAD EN ENTORNOS Y APLICACIONES WEB"** desarrollado por el CCN dentro del Esquema Nacional de Seguridad (ENS).

Enlaces

- https://www.ccn-cert.cni.es/publico/seriesCCN-STIC/series/800-Esquema_Nacional_de_Seguridad/812-Seguridad_en_Entornos_y_Aplicaciones_Web/8I2-Entornos_y_aplicaciones_web-oct11.pdf
- https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29
- http://www.w3schools.com/sql/sql_injection.asp
- https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project
- https://www.owasp.org/index.php/Top_10_2013-Top_10
- http://notasjs.blogspot.com.es/2013/09/politica-del-mismo-origen-same-origin.html