

Seguridad en Sistemas Web

Mikel Egaña Aranguren

mikel-egana-aranguren.github.io

mikel.egana@ehu.eus



Seguridad en Sistemas Web

<https://doi.org/10.5281/zenodo.4302267>

<https://github.com/mikel-egana-aranguren/EHU-SGSSI-01>



Pen-testing

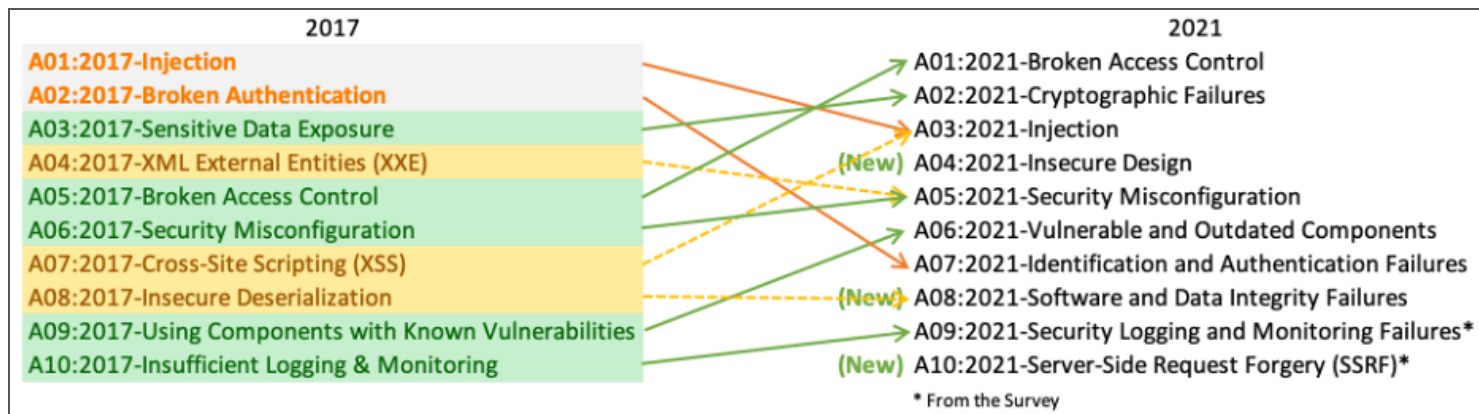
Penetration testing: intentar penetrar en un sistema utilizando las mismas herramientas que un potencial atacante, para descubrir vulnerabilidades y arreglarlas

Principales vulnerabilidades en Sistemas Web

La [Open Web Application Security Project \(OWASP\)](#) analiza las vulnerabilidades más comunes

Informe periódico: OWASP Top Ten (~~2017~~, [2021](#))

Principales vulnerabilidades en Sistemas Web



Common Weakness Enumerations (CWEs)

<https://cwe.mitre.org/>

Principales vulnerabilidades en Sistemas Web

- Rotura de control de acceso
- Fallos criptográficos
- Inyección
- Diseño inseguro
- Configuración de seguridad insuficiente
- Componentes vulnerables y obsoletos

Principales vulnerabilidades en Sistemas Web

- Fallos de identificación y autenticación
- Fallos en la integridad de datos y software
- Fallos en la monitorización de la seguridad
- Server-Side Request Forgery (SSRF)

Rotura de control de acceso

[Broken access control \(A01:2021\)](#)

Rotura de control de acceso

El control de acceso obliga al usuario a actuar solo dentro de los límites establecidos por los permisos definidos

Fallar en el control de acceso resulta en consecuencias graves

Rotura de control de acceso: vulnerabilidades

Violación del principio de "denegación por defecto": el acceso solo debería ser garantizado a ciertas capacidades, roles, o usuarios, pero está al alcance de cualquiera

Criterios de Selección de Guías Docentes:	
Año académico:	2022/23
Centro:	363 - Escuela de Ingeniería de Bilbao
Plan:	GIIGSI30 - Grado en Ingeniería Informática de Gestión y Sistemas de Información
Asignatura:	<input type="text" value=""/> 27706 - Administración de Bases de Datos ▼
Idioma de Grabación:	Castellano ▼
Atras	Buscar

Rotura de control de acceso: vulnerabilidades

Burlar el control de acceso mediante parametros URL o las peticiones APIs

Acceder a una cuenta personal e incluso modificar los datos con el identificador del usuario

Acceso API sin control para los metodos HTTP POST, PUT, y DELETE

Rotura de control de acceso: vulnerabilidades

Elevación de privilegio: actuar como administrador estando logeado como usuario

Replaying o modificación de JSON Web Token (JWT) para elevar privilegios

Mala configuración de CORS permite acceso API desde orígenes no autorizados

Rotura de control de acceso: vulnerabilidades

- [CWE-200: Exposure of Sensitive Information to an Unauthorized Actor](#)
- [CWE-201: Insertion of Sensitive Information Into Sent Data](#)
- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- ...

Rotura de control de acceso: prevención

Denegación por defecto para todos los recursos, excepto públicos

Implementar mecanismos de acceso de control una sola vez y extenderlos a toda la aplicación, minimizando el uso de CORS

No permitir listado de directorios y asegurarse de que no hay archivos de metadatos (ej. Git) ni de backups en el nivel root de la aplicación

Rotura de control de acceso: prevención

Logear todos los intentos de entrada, alertar a los administradores cuando sea necesario (ej. muchos accesos fallidos)

Limitar la tasa de peticiones de API para evitar ataques mediante programas

Rotura de control de acceso: prevención

Identificadores de sesión:

- De servidor (stateful): invalidar al deslogearse
- Stateless JWT tokens: muy poca vida
- Para JWT de tiempo más largo, seguir el protocolo [OAuth](#) de cara a revocar el acceso

Rotura de control de acceso: ejemplo

Llamada a SQL para obtener información de cuenta, con datos sin verificar:

```
01. pstmt.setString(1, request.getParameter("acct"));
```

```
02. ResultSet results = pstmt.executeQuery( );
```

El atacante solo tiene que usar la URL

<https://example.com/app/accountInfo?acct=notmyacct>

Fallos criptográficos

[Cryptographic Failures \(A02:2021\)](#)

Fallos criptográficos

Fallos en los métodos criptográficos usados que resultan en exposición de datos

Determinar las necesidades de protección de los datos tanto en transmisión como en almacenamiento (At rest): leyes de protecciones de datos

Fallos criptográficos

¿Los datos se transmiten como texto plano? Tanto en conexiones externas como internas (ej. servidores de balance de carga): HTTP, SMTP, FTP ...

¿Se usan algoritmos o protocolos obsoletos o débiles?

¿Se usan claves débiles o claves por defecto?

¿Se aplica una gestión de claves adecuada?

Gestión de claves

Ciclo de vida de una clave:

1. Generación
2. Distribución
3. Uso
4. Almacenamiento
5. Rotación, revocación, destrucción

Gestión de claves: generación

Evitar algoritmos débiles

Usar Key Generators o Random Number Generators de alta entropía

Gestión de claves: distribución

Usar conexiones seguras como SSL/TLS para evitar ataques Man in the middle

Gestión de claves: uso

Asegurar que solo las personas autorizadas usan la clave

Gestión de claves: almacenamiento

Idealmente usar hardware dedicado: [Hardware Security Module](#):

- Dejan de funcionar cuando hay un acceso físico
- Cumplen estándares rigurosos como [FIPS 140-2](#) (Nivel 4)

También se pueden usar servicios en la nube: [Google Cloud Key Management](#)

Gestión de claves: rotación

Cuando una clave agota su periodo de validez, se retira, descifrando y volviendo a cifrar los datos con una clave nueva

Cuanto más tiempo esté una clave en circulación, más probable que sea comprometida

Gestión de claves: revocación o destrucción

Si una clave ha sido comprometida, hay que revocarla, aunque este dentro de su periodo de validez

Hay estándares que exigen guardar la clave revocada, por si hiciera falta en el futuro para descifrar datos (ej. en un juicio)

La destrucción completa de la clave implica no poder usarla en el futuro

Gestión de claves: buenas prácticas

No escribir valores de la clave en el código de software, aunque sea seguro

Principio del menor privilegio: un usuario debería tener acceso sólo a las claves que realmente necesita

Usar HSMs

Gestión de claves: buenas prácticas

Automatizar la gestión de claves lo más posible

Dividir la generación, distribución etc. entre diferentes personas

Dividir las claves en fragmentos

Fallos criptográficos

¿El certificado del servidor es válido y de confianza, garantizado por un AC?

¿Se usan las contraseñas de los usuarios directamente como claves, en vez de usar password-based key derivation functions (PBKDFs)?

¿Es la semilla de aleatoriedad usada lo suficientemente fuerte?

Fallos criptográficos: prevención

Clasificar los datos transmitidos, procesados y almacenados según su nivel de exigencia de privacidad

Almacenar la menor cantidad de datos sensibles posible, incluso usando métodos como [Tokenización](#)

Cifrar todos los datos almacenados (Encryption at rest)

Fallos criptográficos: prevención

Usar algoritmos actualizados (No usar MD5, SHA1, ...), claves fuertes, y gestión de claves

Cifrar todos los datos en transito con protocolos como TLS FS (TLS + Forward Secrecy)

No permitir el uso de caches en el cliente para datos sensibles

Fallos criptográficos: prevención

Almacenar contraseñas con hashes + sal

Usar semillas con la mayor entropía posible

Fallos criptográficos: ejemplos

Una aplicación usa la función por defecto de la base de datos para encriptar datos. Sin embargo estos datos se desenscriptan automáticamente al consultarlos, permitiendo el acceso si por ejemplo hay un ataque de Inyección SQL

Fallos criptográficos: ejemplos

Una web no obliga al uso de TLS. Si un atacante monitoriza el tráfico y "baja" una conexión de HTTPS a HTTP puede interceptar consultas y conseguir la cookie de sesión del usuario. El atacante reenvía la cookie (replay) secuestrando la sesión del usuario y obteniendo acceso a los datos de usuario

La base de datos usa hashes sin sal. Un atacante puede pre-calcular los hashes con [GPUs](#)

Fallos criptográficos: vulnerabilidades

- [CWE-259: Use of Hard-coded Password](#)
- [CWE-327: Broken or Risky Crypto Algorithm](#)
- [CWE-331: Insufficient Entropy](#)
- ...

Inyección

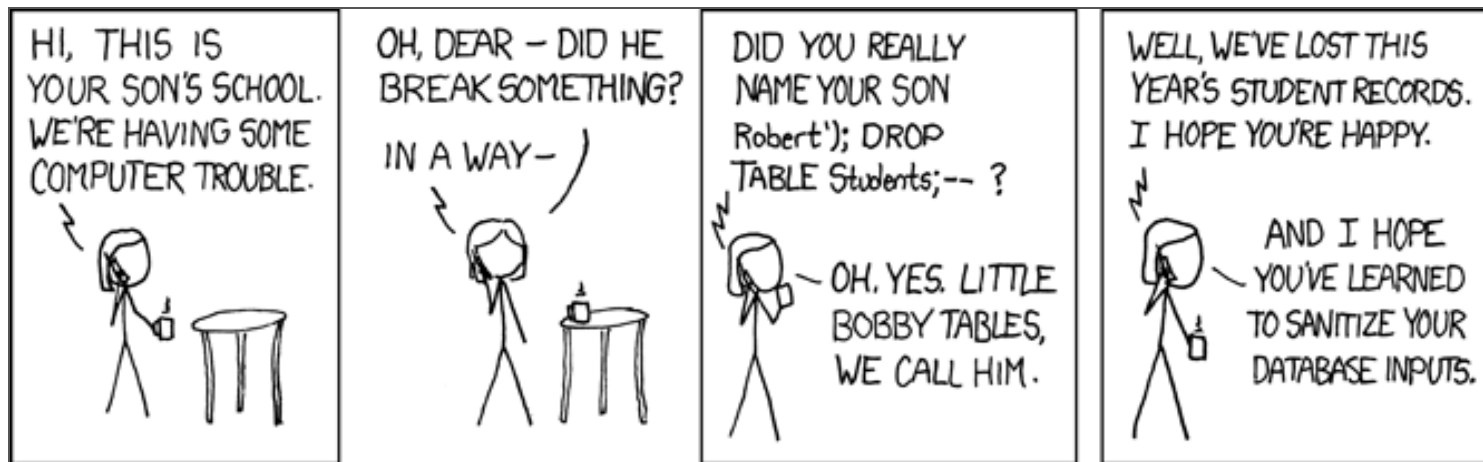
[Injection \(A03:2021\)](#)

Inyección

Una aplicación es vulnerable si:

- Los datos que provee el usuario no se validan, filtran, o limpian
- El intérprete usa directamente consultas dinámicas o llamadas no parametrizadas sin "escaping"
- Datos hostiles introducidos por el usuario son usados directamente. La sentencia SQL o comando que se use contiene estructuras/datos maliciosos en consultas dinámicas, comandos, o procedimientos almacenados

Inyección SQL



Inyección SQL

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

Inyección SQL

Input web: 105 OR 1=1

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

OR 1=1 es siempre true, va a devolver todas las líneas de la columna Users

Consultas parametrizadas

Parametros: valores que se añaden a la consulta al ejecutarla y son analizadas de forma literal (ej. si la variable hace referencia a nombres de columnas, el interprete se asegura de que los parametros son nombres de columnas antes de añadirlas a la consulta y ejecutarla)

Consultas parametrizadas

```
$stmt = $dbh->prepare("INSERT INTO Customers  
(CustomerName,Address,City)  
VALUES (:nam, :add, :cit)");  
$stmt->bindParam(':nam', $txtNam);  
$stmt->bindParam(':add', $txtAdd);  
$stmt->bindParam(':cit', $txtCit);
```

Object-Relational Mapping (ORM)

Frameworks que traducen el contenido de la base de datos a objetos

No hay una conexión directa con la base de datos

Object-Relational Mapping (ORM)

```
String sql = "SELECT ... FROM persons WHERE id = 10"  
DbCommand cmd = new DbCommand(connection, sql);  
Result res = cmd.Execute();  
String name = res[0]["FIRST_NAME"];  
Person p = repository.GetPerson(10);  
String name = p.FirstName;
```

Inyección: detección de vulnerabilidad

Revisión del código fuente

Testeo automatizado de inputs en forma de parametros, cabeceras HTTP, URLs, cookies, JSON, y XML

Application Security Testing en pipelines de [Integración Continua](#)/Despliegue Continuo

Inyección: prevención

Usar APIs seguras que no usen el intérprete. Si no es posible, usar llamadas parametrizadas u ORM

Validar los inputs

"Escapar" todos los caracteres especiales posibles

Usar controles como LIMIT en SQL para evitar perdida masiva de datos

Inyección

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
- CWE-73: External Control of File Name or Path
- ...

Diseño inseguro

[Insecure Design \(A04:2021\)](#)

Diseño inseguro

Fallos de arquitectura, antes de implementar código

Software Assurance Maturity Model

<https://owaspsamm.org/>

[SAMM-v2-PDF.pdf](#)

Diseño inseguro: ejemplos

Una cadena de e-commerce sin protección antibots puede recibir muchas peticiones de compradores a través de bots para vender los productos en otros sitios. Esto se puede evitar, por ejemplo, añadiendo en la arquitectura componentes que detecten bots (Muchas peticiones en poco tiempo, petición al de pocos segundos de publicar un producto, etc.)

Diseño inseguro

- [CWE-209 Generation of Error Message Containing Sensitive Information](#)
- [CWE-256 Unprotected Storage of Credentials](#)
- [CWE-501 Trust Boundary Violation](#)
- [CWE-522 Insufficiently Protected Credentials](#)
- ...

Configuración de seguridad insuficiente

[Security Misconfiguration \(A05:2021\)](#)

Configuración de seguridad insuficiente

La aplicación podría ser vulnerable si:

- Tiene permisos inadecuados en un servicio en la nube
- Hay funciones activas innecesarias (Puertos, servicios, páginas, cuentas, privilegios/permisos)
- Las cuentas por defecto y sus contraseñas siguen activas
- La gestión de los errores desvela información (ej. stack traces en excepciones Java) / los errores son demasiado informativos

Configuración de seguridad insuficiente: prevención

- Sistema de desarrollo automatizado en el que es sencillo desplegar un entorno mas seguro. Los entornos de desarrollo, calidad, y produccion deberian tener exactamente la misma configuracion, a diferencia solo de credenciales. Este proceso deberia estar lo mas automatizado posible para facilitar la creacion de un nuevo entorno seguro lo mas rapido posible
- Un sistema minimalista sin componentes o funciones innecesarias
-

Configuración de seguridad insuficiente

- [CWE-16 Configuration](#)
- [CWE-611 Improper Restriction of XML External Entity Reference](#)
- ...

Componentes vulnerables y obsoletos

[Vulnerable and Outdated Components \(A06:2021\)](#)

Componentes vulnerables y obsoletos

La aplicación podría ser vulnerable si:

- No se saben exactamente todas las versiones de los componentes (Siempre se usa "latest"). Esto incluye componentes que se usan directamente y las dependencias indirectas
- Si el software no tiene soporte, es vulnerable, u obsoleto. Esto incluye al sistema operativo, servidor web, sistema de gestion de bases de datos, APIs y sus componentes, entornos de ejecucions, y librerias

Componentes vulnerables y obsoletos: prevención

Debería haber un proceso de aplicación de parches para:

- Quitar dependencias que no se usan, funcionalidades, componentes, archivos, innecesarios, etc
- Tener un inventario actualizado de componentes en servidor y cliente, incluyendo sus dependencias, de manera automatizada. Visitar regularmente recursos como CVE para encontrar vulnerabilidades en esos componentes

Fallos de identificación y autenticación

Identification and Authentication Failures (A07:2021)

Fallos de identificación y autenticación

La confirmación de identidad de usuarios, autenticación, y la gestión de sesiones son funciones críticas para proteger al sistema contra ataques basados en autenticación

Fallos de identificación y autenticación

Puede haber vulnerabilidades relacionadas con la autenticación si:

- Permite ataque automatizados en los que un atacante tiene una lista de usuarios y contraseñas válidos
- Permite ataques de fuerza bruta
- Permite contraseñas por defecto, débiles o muy conocidas (ej. admin test)
- Usa mecanismos débiles para la recuperación de contraseñas, como el responder ciertas preguntas
- Almacena las contraseñas en texto plano, o con hashes débiles

Fallos de identificación y autenticación: prevención

Puede haber vulnerabilidades relacionadas con la autenticación si:

-

Fallos de identificación y autenticación

- [CWE-297: Improper Validation of Certificate with Host Mismatch](#)
- [CWE-287: Improper Authentication](#)
- [CWE-384: Session Fixation](#)
- ...

Fallos en la integridad de datos y software

[Software and Data Integrity Failures \(A08:2021\)](#)

Fallos en la monitorización de la seguridad

[Security Logging and Monitoring Failures \(A09:2021\)](#)

Server-Side Request Forgery (SSRF)

[Server-Side Request Forgery \(SSRF\) \(A10:2021\)](#)