Fundamentos de ciberseguridad - CIB

ESTUDIO VULNERABILIDADES OWASP

Ciberseguridad

ENRIQUE NIETO LORENZO
12-2-2026

ÍNDICE

1. INTRODUCCIÓN

1.1. Contexto de la Seguridad en el Desarrollo Web

En el panorama actual del desarrollo de software, la seguridad ha dejado de ser una característica opcional o una fase final del despliegue para convertirse en un requisito transversal a todo el Ciclo de Vida de Desarrollo de Software. La creciente complejidad de las aplicaciones web, que han evolucionado desde simples páginas estáticas hacia sistemas dinámicos multicapa con gestión de sesiones y persistencia de datos complejos, ha ampliado exponencialmente la superficie de ataque disponible para actores malintencionados.

La arquitectura de aplicaciones web, especialmente aquellas basadas en el patrón Modelo-Vista-Controlador (MVC), requiere una comprensión profunda no solo de la sintaxis del lenguaje, sino de los flujos de datos y los puntos críticos donde la integridad, confidencialidad y disponibilidad pueden verse comprometidas.

1.2. OWASP y el Estándar de la Industria

El *Open Worldwide Application Security Project* (OWASP) se ha consolidado como la entidad de referencia global en seguridad de software. Funciona como una comunidad abierta y sin ánimo de lucro dedicada a capacitar a las organizaciones para desarrollar aplicaciones fiables. Su metodología se basa en el consenso comunitario, la transparencia y la provisión de documentación gratuita, herramientas y estándares.

El documento más influyente generado por esta organización es el **OWASP Top 10**, una lista actualizada periódicamente que categoriza los riesgos de seguridad más críticos para las aplicaciones web. La versión 2025 de este documento refleja las amenazas más actuales, basándose en datos reales de incidentes y vulnerabilidades detectadas en la industria. Para un Técnico Superior en Desarrollo de Aplicaciones Web, el dominio de estos diez puntos no es solo una cuestión de cumplimiento normativo, sino una competencia fundamental para la ingeniería de software de calidad.

---

2. MARCO TEÓRICO: ANÁLISIS DE LAS 10 VULNERABILIDADES CRÍTICAS (2025)

A continuación, se presenta un desglose conceptual de las diez categorías de riesgo establecidas en la versión 2025, analizando su naturaleza lógica y los principios teóricos para su mitigación.

A01:2025 - Fallos de Control de Acceso (Broken Access Control)

Esta vulnerabilidad ocupa la primera posición debido a su prevalencia y al impacto directo que tiene sobre la privacidad de los datos.

- **Definición Conceptual:** El control de acceso es el mecanismo que garantiza que los usuarios actúen estrictamente dentro de los permisos otorgados. Un fallo en este control ocurre cuando el sistema no verifica adecuadamente si el usuario autenticado tiene la autorización necesaria para acceder a un recurso o ejecutar una acción específica.
- **Mecanismo Lógico:** El error suele residir en la confianza implícita del servidor en los parámetros enviados por el cliente. Si un identificador de objeto (como un ID de usuario en una URL) se utiliza para recuperar datos sin validar que el solicitante es el propietario de dichos datos, se produce una Referencia Directa a Objeto Insegura (IDOR). También incluye la elevación de privilegios, donde un usuario estándar accede a funciones administrativas por falta de validación de roles.
- **Mitigación Teórica:** La defensa principal es el principio de "Denegación por Defecto". A nivel de arquitectura, cada petición HTTP debe pasar por un filtro que valide no solo la identidad (autenticación), sino los permisos específicos sobre el recurso solicitado (autorización) antes de que el controlador procese la lógica de negocio.

A02:2025 - Configuración de Seguridad Incorrecta (Security Misconfiguration)

- **Definición Conceptual:** Vulnerabilidades derivadas de una configuración por defecto, incompleta o mal mantenida en cualquier capa de la pila tecnológica (servidor web, base de datos, framework, etc.).
- **Mecanismo Lógico:** Dejar activados los mensajes de error detallados en producción (revelando la estructura interna del código), mantener cuentas por defecto con contraseñas conocidas, o permitir el listado de directorios en el servidor web. Esto facilita la fase de reconocimiento para un atacante.

- **Mitigación Teórica:** Implementar procesos de "hardening" (endurecimiento) del sistema. Esto implica desactivar todo lo que no sea estrictamente necesario (puertos, servicios, funcionalidades), cambiar credenciales por defecto y configurar el manejo de errores para que sean genéricos de cara al usuario final pero detallados en los registros internos.

A03:2025 - Fallos en la Cadena de Suministro de Software (Software Supply Chain Failures)

Esta categoría ha cobrado gran relevancia debido a la dependencia moderna de librerías de terceros.

- **Definición Conceptual:** Riesgos asociados al uso de componentes externos (librerías, frameworks, módulos) que poseen vulnerabilidades conocidas o que han sido comprometidos en su origen.
- **Mecanismo Lógico:** Si una aplicación utiliza una librería externa desactualizada con una vulnerabilidad crítica, la aplicación hereda dicha vulnerabilidad. Un atacante puede explotar el fallo en el componente para comprometer todo el sistema.
- **Mitigación Teórica:** Mantener un inventario actualizado de todos los componentes de software (SBOM - Software Bill of Materials). Establecer políticas de actualización continua y utilizar herramientas automatizadas para escanear las dependencias en busca de vulnerabilidades conocidas (CVEs).

A04:2025 - Fallos Criptográficos (Cryptographic Failures)

Anteriormente conocida como "Exposición de Datos Sensibles", esta categoría se centra en la raíz del problema: la ausencia o mala implementación de la criptografía.

- **Definición Conceptual:** Se refiere a la protección inadecuada de datos confidenciales, tanto en tránsito (viajando por la red) como en reposo (almacenados en base de datos).
- **Mecanismo Lógico:** Incluye el almacenamiento de contraseñas en texto plano, el uso de algoritmos de hash obsoletos (como MD5 o SHA1) que son vulnerables a ataques de fuerza bruta o tablas *rainbow*, y la transmisión de datos sin cifrado TLS (Transport Layer Security).
- **Mitigación Teórica:** Desde el diseño, se debe clasificar la información según su sensibilidad. Los datos críticos deben cifrarse utilizando algoritmos robustos y estándares actuales. Las contraseñas nunca deben almacenarse directamente; deben transformarse mediante funciones hash unidireccionales con "salt" (valores aleatorios añadidos) y factores de coste computacional para dificultar su reversión.

A05:2025 - Inyección (Injection)

- **Definición Conceptual:** Ocurre cuando datos no confiables suministrados por el usuario son enviados a un intérprete (como el motor de base de datos o el sistema operativo) como parte de un comando o consulta.
- **Mecanismo Lógico:** La aplicación concatena la entrada del usuario directamente con la cadena de comandos del sistema. El intérprete, al no distinguir entre los datos y las instrucciones, ejecuta el código malicioso insertado por el atacante. El caso más paradigmático es la Inyección SQL, donde se manipulan las consultas a la base de datos.
- **Mitigación Teórica:** La solución teórica es la separación estricta entre datos y comandos. Esto se logra mediante el uso de interfaces parametrizadas (consultas preparadas), donde el intérprete trata la entrada del usuario estrictamente como datos literales y nunca como código ejecutable.

A06:2025 - Diseño Inseguro (Insecure Design)

Esta categoría subraya que la seguridad no es solo una cuestión técnica, sino arquitectónica.

- **Definición Conceptual:** Representa fallos que provienen de la ausencia de controles de seguridad en la fase de diseño. A diferencia de un error de implementación (un "bug" en el código), un diseño inseguro implica que, aunque el código se escriba perfectamente según las especificaciones, el sistema sigue siendo vulnerable porque la especificación misma era deficiente.
- **Mecanismo Lógico:** Ejemplos incluyen flujos de recuperación de contraseñas basados en "preguntas de seguridad" (cuya respuesta es información pública) o lógica de negocio que no prevé el abuso de funcionalidades (como la compra masiva de artículos por bots).
- **Mitigación Teórica:** Se requiere implementar un ciclo de vida de desarrollo seguro (SDLC) que incluya el modelado de amenazas en las fases iniciales. Se debe adoptar la "Seguridad por Diseño" y "Seguridad por Defecto" como principios rectores de la arquitectura del software.
- **Distinción fundamental:** Fallo de Diseño frente a Defecto de Implementación. Para comprender esta categoría es esencial distinguir entre ambos conceptos, ya que esta es la única vulnerabilidad del Top 10 que se origina exclusivamente en la fase de diseño y no en la de codificación. Un defecto de implementación se produce cuando un diseño seguro se programa incorrectamente: el requisito especificaba el uso de HTTPS, pero el certificado no se renovó. La solución es corregir el código o la configuración específica. Un fallo de diseño, en cambio, implica que el control de seguridad nunca se contempló en la especificación. Aunque el código se implemente de forma impecable según las

especificaciones, el sistema permanece vulnerable porque la especificación misma es deficiente. La consecuencia práctica de esta distinción es que un fallo de diseño no puede resolverse mediante parches o correcciones de código: requiere un rediseño arquitectónico de la funcionalidad afectada.

- **Consecuencias específicas del Diseño Inseguro:** La ausencia de controles de seguridad en la fase de diseño genera cinco categorías principales de riesgo.

    1. Sin un modelado de amenazas previo al desarrollo, las vulnerabilidades permanecen latentes hasta la fase de producción, donde su corrección implica un coste significativamente mayor.

    2. Si la arquitectura no contempla una separación de roles desde el inicio, la elevación de privilegios se vuelve trivial para un atacante.

    3. Cuando la lógica de negocio no incorpora validaciones de abuso, las funcionalidades pueden ser explotadas de formas no previstas, como la aplicación ilimitada de cupones de descuento o la introducción de valores negativos en sistemas de pujas.

    4. La ausencia de limitación de tasa de peticiones (rate limiting) en el diseño permite que los ataques de fuerza bruta se ejecuten sin restricción.

    5. Los mensajes de error diferenciados entre tipos de fallo, cuando no se consideran desde la fase de diseño, facilitan la enumeración de usuarios válidos del sistema.

- **La Orientación a Objetos como decisión de diseño seguro:** Un aspecto frecuentemente subestimado de esta vulnerabilidad es el papel que desempeña la arquitectura orientada a objetos como mecanismo de seguridad por diseño. La encapsulación de atributos mediante modificadores de acceso privados no es únicamente una práctica de ingeniería del software: constituye una barrera arquitectónica que impide el acceso directo a datos sensibles desde capas externas. De forma análoga, la centralización del acceso a datos en clases específicas de persistencia garantiza que toda interacción con la base de datos se canalice a través de métodos controlados, eliminando por diseño la posibilidad de inyección SQL ad-hoc. Estas decisiones arquitectónicas, tomadas antes de escribir código funcional, representan exactamente lo que OWASP denomina "patrones de diseño seguros" y constituyen la diferencia entre un sistema que necesita parches constantes y uno que es inherentemente resistente.

- **CWEs asociados:** Las debilidades más representativas vinculadas a esta categoría incluyen CWE-256 (almacenamiento de credenciales sin protección por diseño), CWE-269 (gestión inadecuada de privilegios), CWE-434 (subida de ficheros sin restricciones de tipo) y CWE-522 (credenciales protegidas de forma insuficiente).

A07:2025 - Fallos de Identificación y Autenticación (Authentication Failures)

- **Definición Conceptual:** Debilidades en los mecanismos que verifican la identidad del usuario y gestionan su sesión activa.
- **Mecanismo Lógico:** Permitir contraseñas débiles, no contar con protección contra ataques de fuerza bruta (intentos ilimitados de inicio de sesión), o una mala gestión de los identificadores de sesión (por ejemplo, permitir que un ID de sesión sea reutilizado o no invalidarlo tras el cierre de sesión).
- **Mitigación Teórica:** Implementar autenticación multifactor (MFA), establecer políticas de complejidad de contraseñas, limitar la tasa de intentos fallidos (Rate Limiting) y asegurar que la gestión de sesiones utilice identificadores aleatorios, de vida corta y que se invaliden correctamente.

A08:2025 - Fallos en la Integridad del Software y Datos (Software or Data Integrity Failures)

- **Definición Conceptual:** Fallos relacionados con la falta de verificación de la integridad de los datos o el código proveniente de fuentes externas.
- **Mecanismo Lógico:** Un ejemplo crítico es la deserialización insegura, donde la aplicación reconstruye objetos a partir de datos serializados sin verificar su contenido, permitiendo la ejecución remota de código. También incluye la descarga de actualizaciones de software sin verificar su firma digital.
- **Mitigación Teórica:** Verificar siempre la firma digital de los componentes de software. En cuanto a los datos, evitar la serialización de objetos complejos provenientes del cliente o implementar controles de integridad estrictos.

A09:2025 - Fallos en el Registro y Monitorización (Security Logging & Alerting Failures)

- **Definición Conceptual:** La incapacidad de detectar, registrar y responder a incidentes de seguridad en tiempo real.
- **Mecanismo Lógico:** Si los eventos críticos (como fallos de login, errores de acceso o errores de sistema) no se registran, o si los registros se almacenan localmente sin copias de seguridad, es imposible detectar un ataque en curso o realizar un análisis forense posterior.

- **Mitigación Teórica:** Asegurar que todos los eventos de seguridad generen registros con contexto suficiente. Implementar sistemas centralizados de monitorización y alertas que avisen a los administradores ante patrones de comportamiento anómalos.

A10:2025 - Manejo Inadecuado de Condiciones Excepcionales (Mishandling of Exceptional Conditions)

Una nueva categoría que enfatiza la robustez lógica del código.

- **Definición Conceptual:** Ocurre cuando la aplicación no gestiona correctamente situaciones imprevistas (errores de red, base de datos caída, entradas malformadas), dejando al sistema en un estado vulnerable o inconsistente.
- **Mecanismo Lógico:** Un ejemplo es el principio de "Fail Open" (fallar abierto), donde un error en el mecanismo de seguridad permite el acceso en lugar de bloquearlo. También incluye la revelación de trazas de pila (stack traces) completas al usuario.
- **Mitigación Teórica:** Aplicar el principio de "Fail Secure" (fallar seguro) o "Fail Closed". Cualquier excepción debe ser capturada y manejada de forma que el sistema deniegue el acceso o la operación por defecto, registrando el error internamente y mostrando un mensaje genérico al usuario.

---

3. ANÁLISIS DE CASO: APLICACIÓN "LOGIN/LOGOFF MULTICAPA"

En este apartado, se aplicarán los conceptos teóricos desarrollados anteriormente al análisis de la arquitectura de referencia del curso: la aplicación **2XXDWESLoginLogoff**. Este análisis se basa en la documentación técnica y los diagramas de arquitectura proporcionados en el material del curso.

3.1. Descripción Arquitectónica del Sistema

La aplicación objeto de estudio es un sistema web desarrollado en PHP siguiendo el patrón de diseño **Modelo-Vista-Controlador (MVC)**. Su estructura de directorios separa claramente las responsabilidades:

- **Modelo (/model):** Encapsula la lógica de negocio y el acceso a datos. Destacan las clases Usuario (entidad) y UsuarioPDO (lógica de acceso a datos), así como la interfaz DB y su implementación DBPDO.
- **Controlador (/controller):** Gestiona el flujo de la aplicación. Existen controladores específicos para cada vista, orquestados por un punto de entrada único (index.php).
- **Vista (/view):** Responsable de la presentación. Incluye vistas públicas (vInicioPublico, vLogin, vRegistro) y privadas (vInicioPrivado, vMiCuenta), todas heredando de una estructura común definida en Layout.php.
- **Configuración (/config):** Contiene ficheros críticos como confDB.php y confAPP.php.

Esta arquitectura multicapa es, en sí misma, una primera medida de seguridad al promover la separación de intereses, pero introduce puntos específicos que deben ser asegurados teóricamente.

3.2. Análisis de Vulnerabilidades en el Ciclo de Autenticación y Sesión

A07: Identificación y Autenticación en UsuarioPDO y cLogin

El núcleo de la seguridad de la aplicación reside en el proceso de login. Según el diagrama de navegación, el flujo comienza en vInicioPublico, pasa a vLogin y, si la autenticación es exitosa, deriva a vInicioPrivado.

Desde el punto de vista teórico, el método UsuarioPDO::validarUsuario() es crítico.

1. **Protección contra Fuerza Bruta:** La base de datos incluye el campo T01_NumConexiones. Teóricamente, este campo podría utilizarse no solo para estadística, sino para implementar una lógica de bloqueo tras un número determinado de intentos fallidos en un tiempo corto, mitigando ataques de fuerza bruta.
2. **Gestión de Sesión:** Una vez validado el usuario en el modelo, el controlador debe iniciar una sesión. Es vital que el identificador de sesión se regenere en este instante preciso (prevención de *Session Fixation*). Además, el campo T01_FechaHoraUltimaConexion debe actualizarse para permitir controles de inactividad (Timeout), cerrando la sesión automáticamente si el tiempo transcurrido excede un límite seguro.

A04: Fallos Criptográficos en el Modelo de Datos

El modelo físico define la tabla T01_Usuario con un campo T01_Password.

- **Almacenamiento:** Teóricamente, bajo ninguna circunstancia este campo debe almacenar la contraseña en texto claro. La clase UsuarioPDO, en sus métodos de altaUsuario() y validarUsuario(), debe implementar algoritmos de hashing robustos (como bcrypt o Argon2). Si la aplicación solo usara MD5 (un algoritmo obsoleto), estaría incurriendo en un fallo criptográfico grave (A04).
- **Tránsito:** Toda la comunicación entre las vistas (vLogin, vRegistro) y el servidor debe realizarse sobre HTTPS para evitar la interceptación de credenciales.

3.3. Riesgos en la Gestión de Datos e Inyección

A05: Inyección SQL en la Abstracción de Datos (DBPDO)

La aplicación utiliza una clase envoltorio para la base de datos: DBPDO. El método estático ejecutaConsulta(entrada sentenciaSQL, entrada parametros) es la pieza clave para prevenir la inyección SQL.

- **Fundamento Teórico:** Si DBPDO estuviera implementado simplemente concatenando la variable sentenciaSQL con los valores de entrada, la aplicación sería vulnerable. Sin embargo, la firma del método incluye un argumento explícito entrada parametros. Esto indica conceptualmente el uso de **Consultas Preparadas**.
- **Aplicación:** Al separar la estructura SQL de los datos, el motor de base de datos trata los parámetros como valores literales. Así, aunque un usuario malintencionado introduzca código SQL en el formulario de login, este será tratado como una cadena de texto inofensiva y no como una instrucción ejecutable.

A08: Integridad en la Serialización de Objetos

El objetivo del tema menciona "Serialización automática en la sesión". En PHP, cuando se guarda un objeto en la sesión ($_SESSION), este se serializa. Si la aplicación permitiera que datos no confiables (como una cookie manipulada por el usuario) fueran deserializados directamente en el contexto de la aplicación sin validación, se estaría vulnerando la integridad del software (A08). La arquitectura debe garantizar que solo los objetos generados internamente por el servidor (como la instancia de la clase Usuario tras el login) sean almacenados y recuperados de la sesión.

3.4. Control de Acceso y Lógica de Negocio

A01: Control de Acceso en el Front Controller

El diagrama de navegación muestra una clara distinción entre zonas públicas y privadas. La transición de vInicioPublico a vInicioPrivado o vMiCuenta debe estar protegida por un control de acceso riguroso.

- **Lógica de Controlador:** Cada controlador que gestione una vista privada (ej. cMiCuenta, cInicioPrivado) debe verificar teóricamente dos cosas antes de renderizar la vista:
    1. **Autenticación:** ¿Existe una sesión válida activa?
    2. **Autorización:** ¿Tiene el usuario el perfil adecuado? (El modelo de datos contempla un campo T01_Perfil). Si estas comprobaciones faltan o se confía únicamente en ocultar los enlaces en la vista (Layout.php), un atacante podría acceder directamente invocando la URL del controlador privado (Bypass de Control de Acceso).

A02: Configuración de Seguridad en la Estructura de Ficheros

La estructura de directorios define carpetas como /config, /core, /model.

- **Riesgo:** Si el servidor web no está configurado correctamente, un usuario podría solicitar directamente el archivo /config/confDB.php. Aunque PHP procesa el archivo, si hay un error de configuración, podría servirse como texto plano, revelando las credenciales de la base de datos.
- **Mitigación Arquitectónica:** La teoría dicta que estas carpetas deben estar fuera del directorio raíz público (public_html o webroot) o protegidas mediante directivas de servidor (como archivos .htaccess en Apache) que denieguen el acceso directo a cualquier recurso que no sea el punto de entrada index.php.

A06: Diseño Inseguro en la Arquitectura del Sistema

La aplicación presenta decisiones de diseño que mitigan esta categoría de forma significativa, junto con carencias puntuales que deberían abordarse.

- **Medidas de diseño seguro presentes**: La arquitectura del sistema incorpora varios patrones que OWASP identifica como mitigaciones directas de A06. En primer lugar, las clases del modelo (Usuario, Departamento) implementan encapsulación estricta con atributos privados, lo que impide que las capas de controlador o vista manipulen directamente datos sensibles como contraseñas o perfiles. En segundo lugar, la centralización del acceso a base de datos a través de la clase DBPDO y su método

ejecutaConsulta() con parámetros obliga a que toda nueva funcionalidad utilice sentencias preparadas, eliminando la inyección SQL por diseño arquitectónico y no por disciplina individual del programador. En tercer lugar, el controlador frontal (index.php) actúa como punto único de verificación de sesión y autorización, implementando el principio de denegación por defecto: el array de configuración en confAPP.php define una lista blanca de páginas públicas, de modo que cualquier nueva vista es privada por defecto. Finalmente, las clases transversales ValidaForms y ErrorApp funcionan como la librería de patrones seguros que OWASP recomienda, centralizando la validación de entrada y el control de fuga de información respectivamente.

- **Carencias de diseño identificadas:** El análisis revela tres aspectos donde el diseño no contempló controles de seguridad. Primero, el flujo de autenticación en cLogin no incorpora un mecanismo de limitación de tasa (rate limiting), lo que permite un número ilimitado de intentos de inicio de sesión. Aunque el campo T01_NumConexiones existe en el modelo de datos, su uso como contador de bloqueo no fue diseñado en la especificación funcional. Segundo, la regeneración del identificador de sesión tras la autenticación exitosa (prevención de session fixation) debe contemplarse como requisito de diseño, no como detalle de implementación. Tercero, el flujo de registro de usuarios revela información sobre la existencia de cuentas mediante mensajes diferenciados, un compromiso entre usabilidad y seguridad que debería haberse resuelto en la fase de diseño mediante alternativas como la verificación por correo electrónico.

- **Valoración del grado de afectación:** El nivel de exposición a A06 puede calificarse como medio-bajo en términos arquitectónicos, dado que las decisiones estructurales del sistema (OOP, MVC, controlador frontal centralizado) proporcionan una base sólida. Las carencias identificadas son controles puntuales que pueden incorporarse sin requerir un rediseño de la arquitectura, lo que confirma que el diseño base del sistema es fundamentalmente seguro.

3.5. Gestión de Errores y Monitorización

A10 y A09: El rol de cError y vError

La arquitectura incluye explícitamente un controlador y una vista de error (cError, vError).

- **Manejo de Excepciones (A10):** Teóricamente, cualquier excepción no capturada en la lógica de negocio (como un fallo de conexión en DBPDO) debe ser redirigida a cError. Es crucial que vError muestre un mensaje amigable y genérico al usuario ("Ha ocurrido un error, inténtelo más tarde") y no la traza del error de PHP o SQL, para evitar la fuga de información técnica.

- **Registro (A09):** Simultáneamente, el sistema debe registrar internamente el detalle técnico del error (fecha, usuario, traza, fichero) en un log del servidor. Sin este registro, la detección de intentos de ataque o fallos recurrentes es imposible.

---

4. CONCLUSIONES

El análisis teórico de la arquitectura de la aplicación bajo el prisma del OWASP Top 10 2025 revela que la seguridad no es un componente que se añade al final del desarrollo, sino que es intrínseca a la arquitectura del software.

1. **La Arquitectura como Defensa:** La elección de un modelo MVC multicapa, la encapsulación de datos sensibles en clases con atributos privados y el uso de abstracciones para la base de datos (DBPDO) proporcionan una estructura robusta que facilita la implementación de defensas contra inyecciones (A05) y constituye en sí misma una mitigación del diseño inseguro (A06), al aplicar patrones de seguridad por diseño desde la fase arquitectónica.
2. **La Importancia de la Sesión:** La correcta gestión del ciclo de vida de la sesión y la autenticación (UsuarioPDO) es el pilar fundamental que previene el robo de identidad (A07) y el acceso no autorizado (A01).
3. **Defensa en Profundidad:** Ningún control es suficiente por sí solo. La seguridad de la aplicación depende de la combinación de validaciones en el controlador, uso correcto de criptografía en el modelo, configuración segura del servidor y una gestión adecuada de errores. Todo ello debe nacer en la fase de diseño y no incorporarse como medida correctiva posterior, lo que convierte a A06: Diseño inseguro en la categoría transversal que condiciona la eficacia de todas las demás.