😟 SEMINARIO 5
VALIDACIÓN DE ENTRADAS Y TÉCNICAS DE DESARROLLO SEGURO

Estrategias para rechazar datos inválidos

PROYECTO "*S-81 ISAAC PERAL*" V4.2

¿LOCALIZASTE ALGUNA ERRATA O PROBLEMA? POR FAVOR, NOTIFÍCALA A JOSÉ MANUEL REDONDO LÓPEZ (REDONDOJOSE@UNIOVI.ES) ¡GRACIAS POR AYUDARNOS A MEJORAR LA ASIGNATURA!

Departamento de Informática
Universidad de Oviedo
Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

Acerca del uso de contenido generado por IA

- **En esta presentación se usan algunas imágenes generadas por IA**
  - Salvo error, **cualquier imagen a la que no se le atribuya una fuente** u origen expreso
- **La IA generativa usada para ello es Microsoft Copilot**
  - https://copilot.microsoft.com/
- **Se ha restringido el uso de estas imágenes a la ilustración de los conceptos explicados en algunas de las páginas**
  - Es decir, **como refuerzo visual** a lo explicado en algunas transparencias
  - El procedimiento ha sido describirle a la IA con toda la precisión posible los elementos que quería que apareciesen en la imagen (prompt)
    - Y la selección del mejor resultado obtenido, a juicio del autor de esta presentación
    - **No se ha mencionado ni indicado que se copie el estilo a ningún autor, ni que se plagien obras concretas**
- **El autor declara expresamente su apoyo al trabajo de los artistas, ilustradores y creadores, de extrema importancia en la actualidad**
  - El uso de estas técnicas se ha hecho solo con fines de mejora de las explicaciones, y cuando la alternativa era **no contar con refuerzos visuales** por restricciones de tiempo y presupuesto

¿Qué es este seminario?

- **En este seminario vamos a dar ejemplos prácticos de cómo hacer tareas de seguridad vinculadas a la fase de implementación del SSDLC**
  - Como complemento al **Tema 6** de teoría
  - Muchos en forma de **checklist** que permitan comprobar fácilmente si se han implementado
- **Estos ejemplos se muestran a propósito en tecnologías que se usan en otras asignaturas del grado para favorecer la integración entre asignaturas**
  - Node y Spring Boot (SDI) y PHP (SEW)
- **También mencionaremos conceptos de**
  - TPP: Manejo de concurrencia
  - DLP/AMD: Manejo y validación de expresiones regulares
- **El checklist completo en formato editable se puede descargar de aquí:**
  - https://github.com/jose-r-lopez/SSI_Extra_Materials/wiki/5.-%E2%9C%85-Secure-Coding-Checklist
  - Se trata de una **simplificación del ASVS** (**Tema 6**), pensada para tu TFG o en futuros proyectos de software como preformación para utilizar el enfoque ASVS

ÍNDICE

< Ir al Índice

Requisitos de Seguridad

En el formato dado en Ingeniería de Requisitos (IR)

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Ejemplo de Requisitos de Seguridad

Seguridad de Sistemas Informáticos

- **Esta es una lista de posibles requisitos de seguridad**
  - Se pueden usar como base para hacer la tuya
  - Adaptarla a tu aplicación, propósito, usuarios y restricciones
  - Considérala una lista de "entrenamiento" para software no crítico
- **Durante este seminario, te daremos una checklist de implementación**
  - ¡Así podrás hacer un seguimiento fácil de lo hecho para cumplir con cada requisito!
- **Una vez tengas soltura con ella debes usar el ASVS de teoría**

- **RS1 Validación**
  - RS1.1 **No confiar en nadie**: Validar y sanear (en casos especiales) todos los datos, incluidos los de la propia BBDD
  - RS1.2 **Hacer toda la validación en el servidor**
    - Priorizando el uso de listas de permitidos
  - RS1.3 **Codificar** todas las salidas
- **RS2 Cifrado**
  - RS2.1 **Cifrar todos los datos guardados**
  - RS2.2 **Cifrar todos los datos "en tránsito"**
    - Ej.: enviados/recibidos por el usuario, BBDD, APIs, etc.
  - RS2.3 **Hashear e incorporar bits de sal** (al menos 28 caracteres) a todas las passwords (o no guardarlas)
  - RS2.4 **HTTPs obligatorio** para todo el sitio web
  - RS2.5 Usar la **última versión de TLS** para cifrado
- **RS3 Gestión de dependencias**
  - RS3.1 **Escanear las librerías y componentes de 3ºs (SCA)**
    - Para encontrar vulnerabilidades conocidas antes de usarlos
    - Y también periódicamente una vez incorporados
  - RS3.2 **Minimizar la superficie de ataque**
    - Incluyendo el menor nº de dependencias posible

Ejemplo de Requisitos de Seguridad

RS4 Gestión de datos

- RS4.1 **Clasificar y etiquetar** todos los datos (guardados, recopilados, creados...) de la aplicación
  - Según su **criticidad**
- RS4.2 Guardar todos los secretos de la aplicación en una **secret store**
- RS4.3 Prohibir el **"hardcoding"** de secretos
- RS4.4 **No poner datos sensibles** en comentarios
  - Cadenas de conexión, claves, etc.
- RS4.5 **Los mensajes de error nunca** contienen información técnica
- RS4.6 **Usar solo parametrized queries** para el acceso a datos
- RS4.7 No pasar datos importantes en los parámetros de una URL

RS5 Tecnológicos

- RS5.1 Usar todas las cabeceras de seguridad aplicables y **CSP (Seminario 6)**
- RS5.2 Usar una **configuración de cookies segura**
- RS5.3 Usar **características de seguridad existentes**
  - Ej.: De los frameworks elegidos
- RS5.4 Usar **la última versión** estable, soportada y actualizada de los frameworks
- RS5.5 Mantener las **dependencias actualizadas**
- RS5.6 Usar **herramientas de seguridad** (SCA, SAST...)
  - Antes de la puesta en producción de la aplicación
- RS5.7 Hacer code review de la aplicación
  - Antes de que sea puesta en producción
- RS5.8 Captura todos los errores y trata los fallos
  - Sin caer en estados inesperados
- RS5.9 Deshabilita la caché en páginas que contienen información privada (Ej.: passwords)

Ejemplo de Requisitos de Seguridad

RS6 Cuentas de usuario

- RS6.1 Todas las cuentas de la aplicación son **de servicio**
  - Nunca cuentas de usuario, y una cuenta única por aplicación
- RS6.2 Usar password managers y **prohibir reutilizar claves**
- RS6.3 Forzar el **uso de claves largas** (pass phrases)
  - Con un mínimo de complejidad
- RS6.4 Verificar que las claves no hayan sido **filtradas**
  - Usando un servicio externo que recopile filtraciones
- RS6.5 **Activar MFA** en todas las cuentas importantes
- RS6.6 **No forzar el cambio periódico de claves**
  - Solo tras descubrirse una brecha
- RS6.7 Asignar **roles** a los usuarios y los permisos asociados a esos roles
- RS6.8 Fijar **un único método de autenticación** y gestión de identidades para toda la aplicación
  - **Evita guardar claves** (usa un servicio de autenticación de 3ºs)
- RS6.9 Forzar el **mínimo privilegio** para todas las cuentas
- RS6.10 Habilitar copiar / pegar en los **elementos del GUI**
  - Permite usar password managers
  - Y deshabilita autocompletar en las entradas de claves

RS7 Funcionalidades particulares

- RS7.1 Hacer **threat modeling** de la aplicación
- RS7.2 Evitar la **subida de archivos**
  - **Si no es posible,** sigue estas recomendaciones
  - https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html

RS8 Log

- RS8.1 **Registrar todos los intentos de acceso**
- RS8.2 **No incluir información sensible** en el log
- RS8.3: Usar un formato de log **compatible** con tus herramientas de análisis (SIEMs...)
- RS8.4: Enviar los logs a un **almacén central**
  - Mejora el análisis y la correlación de los mismos
- RS85: Haz log de **intentos de acceso inválidos**
  - Detecta intentos maliciosos (Ej.: entradas maliciosas) con código (**defensa proactiva**)
  - **Genera alertas de seguridad** si una aplicación está bajo ataque

< Ir al Índice

✅ Validación de Entradas y Codificación de Salidas

Formas prácticas de tratar la información de entrada o salida

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Validación de Entradas

Seguridad de Sistemas Informáticos

- **La entrada de los usuarios / archivos / APIs externas debe tratarse con precaución**
  - Uno de los vectores de ataque principales
- **Las entradas deben validarse**
  - Con API centralizadas probadas y de acuerdo con su contexto
- **La validación de entradas debe**
  - Aplicarse a todos los datos
  - Definir un conjunto permitido de caracteres aceptados
  - Definir una longitud mínima y máxima
    - Dígitos decimales y no decimales, el tamaño máximo de los nombres...
  - Definir un tipo o rango esperado

1. Lista de verificación de validación de entradas	
Garantizar el cumplimiento de los principios obligatorios de validación de entradas	
	Realizar toda la validación de datos en un sistema de confianza (por ejemplo, el servidor)
	Identificar todas las fuentes de datos y clasificarlas en confiables y no confiables. Validar todos los datos de fuentes que no sean de confianza (por ejemplo, bases de datos, archivos, etc.)
	Debe haber una lógica de validación de entradas centralizada única para la aplicación
Realizar una codificación de datos adecuada	
	Determinar si el sistema admite conjuntos de caracteres extendidos UTF-8 y, de ser así, validar después de que se complete la decodificación UTF-8
	Codificar datos en un conjunto de caracteres común antes de validar (Canonicalizar)
	Especificar los conjuntos de caracteres adecuados, como UTF-8, para todas las fuentes de entrada
	Comprobar que los valores de las cabeceras HTTP, tanto en las solicitudes como en las respuestas contengan solo caracteres ASCII
Implementar una gestión segura del contenido de los datos	
	Si se debe permitir algún carácter potencialmente peligroso como entrada, asegurarse de implementar controles adicionales como la codificación de salidas, uso seguro de API específicas para ciertas tareas y hacer contabilidad del uso de esos datos en toda la aplicación. Ejemplos de caracteres peligrosos comunes incluyen: < > " ' % ( ) & + \\' \"
	Si la rutina de validación estándar no puede manejar las siguientes entradas, deben verificarse manualmente Comprobar si hay bytes nulos (%00) Comprobar si hay caracteres de línea nueva (%0d, %0a, \r, \n) Comprobar si hay caracteres "punto-punto-slash" (.. / o .. \) que puedan cambiar una ruta. En los casos en que se admita la codificación de conjuntos de caracteres extendidos UTF-8, incluir las representaciones alternativas como: %c0%ae%c0%ae/ (Usar la canonicalización para evitar la doble codificación u otras formas de ataques de ofuscación)
	Validar todas las entradas con una lista "blanca" de caracteres permitidos, siempre que sea posible
Validar tipos de datos y límites	
	Validar la longitud de los datos
	Validar el rango de datos
	Validar sobre los tipos de datos esperados
Realizar la validación de datos de forma segura	
	Todos los errores de validación deben causar que la entrada se rechace
	Validar todos los datos proporcionados por el cliente antes de su procesamiento, incluidos todos los parámetros, URLs y contenido de cabeceras HTTP (por ejemplo, nombres y valores de cookies). Asegurarse de incluir las respuestas automatizadas de JavaScript u otro código incrustado
	Validar datos de redirecciones (un atacante puede enviar contenido malicioso directamente al objetivo de la redirección, eludiendo así la lógica de la aplicación y cualquier validación realizada antes de dicha redirección)

ESTRATEGIAS DE VALIDACIÓN DE DATOS

- **Los datos que provienen de una entidad externa nunca deben ser de confianza**
  - *"All input is evil"* – Michael Howard (*"Writing Secure Code"*)
- **La validación de datos debe ser sintáctica y semántica**
  - **Sintáctica** obliga a los datos a tener sintaxis y estructura correctas: Fechas, moneda, DNIs, tfnos...
  - **Semántica** obliga a los valores a ser correctos en un contexto o según unas reglas de negocio
    - La fecha de inicio debe ser anterior a la fecha de finalización, el precio de un artículo está dentro de un rango admitido, el tipo de vía residencial está dentro de los tipos admitidos (calle / plaza /...) ...
- **Si estas cosas se comprueban lo antes posible, las entradas no autorizadas se pueden detectar rápidamente y actuar en consecuencia**
  - De forma planificada y exhaustiva, como parte de los requisitos del sistema
  - Si es un caso claro de manipulación maliciosa, se deben **implementar respuestas (no solo descartar los datos) en el software** (suspensión, bloqueo...) y lanzar alertas de seguridad (ver teoría)
- **Siempre favoreciendo lista de elementos permitidos (el resto está bloqueado) sobre listas de bloqueo (el resto de los elementos se permiten)**

LISTA DE PERMITIDOS VS LISTA DE BLOQUEADOS

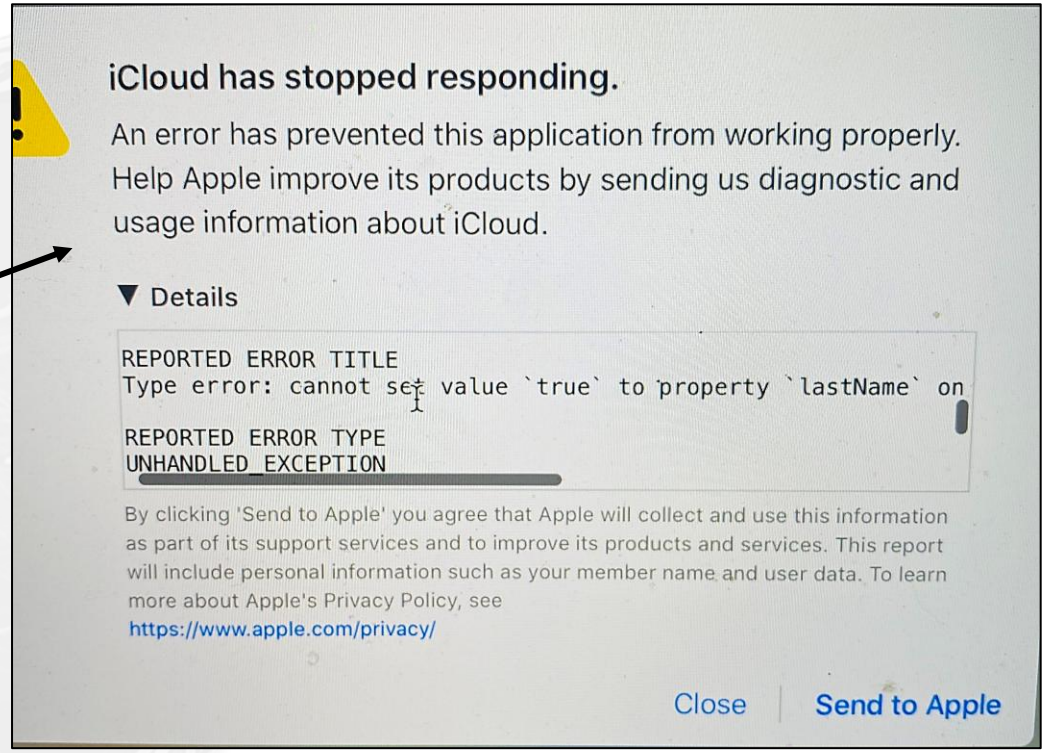
- **Usar listas de bloqueo cuando el conjunto de elementos a bloquear es finito, conocido en su totalidad, y no muy grande**
  - **Bloqueo de clientes / redes específicas** en entornos con un nº restringido/conocido de ellos (LAN)
  - **Bloqueos regionales** (por país/zona): países conocidos por ser fuente de un gran número de ataques
  - **Conexiones de clientes desde la red ToR**: Su número de nodos de salida es limitado y conocido
    - Se pueden bloquear individualmente (https://www.dan.me.uk/torlist/?exit)
  - **Bloqueo de dominios conocidos como maliciosos**: https://github.com/StevenBlack/hosts
  - **Bloqueos de páginas específicas por diferentes razones**, si el número de ellas es limitado
- **Validar con lista de permitidos suele ser más adecuado para entradas de usuario**
  - Se trata de **definir exactamente lo que está autorizado**
    - Cualquier cosa que no esté en esa definición **NO ESTÁ AUTORIZADA**
  - En el caso de **datos bien estructurados** (fechas, números de seguridad social, códigos postales, correos electrónicos...) define un patrón de validación muy fuerte, **generalmente con exp. regulares**
    - Suelen estar **predefinidas y estandarizadas para tipos de datos comunes**
  - Además, si la entrada viene de un **conjunto fijo de opciones**, como una lista desplegable, la entrada **debe coincidir exactamente con uno de los valores ofrecidos al usuario**

IMPLEMENTANDO ESTRATEGIAS DE VALIDACIÓN DE ENTRADA

- **La validación de entradas se puede implementar con cualquier técnica que garantice la corrección sintáctica y semántica; por lo general es una de estas**
  - **Validadores nativos** de cualquier framework moderno usado para crear el sitio web
    - Django validators: https://docs.djangoproject.com/en/2.2/ref/validators/
    - Apache Commons validators: https://commons.apache.org/proper/commons-validator/
    - PHP (SEW): https://www.w3schools.com/php/php_form_validation.asp
    - Node.Js (SDI): https://www.npmjs.com/package/validator?activeTab=readme
    - Spring Boot (SDI): https://reflectoring.io/bean-validation-with-spring-boot/
    - ...
  - **Validación con esquemas JSON y XML (XSD)** para las entradas que siguen estos formatos
  - **Convertir tipos** (Ej. `Integer.parseInt()` en Java, `int()` en Python...) y **control de excepciones**
  - **Comprobar el valor mínimo y máximo** para parámetros y fechas numéricos, y **Longitud** para cadenas
  - **Comprobación estricta de un conjunto de valores permitidos**: especialmente para parámetros de cadena que sólo pueden tener un pequeño número de valores posibles (días de la semana, meses...)
  - **Expresiones regulares** para cualquier otro dato estructurado
    - Se debe validar toda la cadena de entrada, **evitando el uso de comodines**

La conversión de tipos es importante

- **Validar sintáctica / semánticamente no es suficiente: no ignores la conversión de tipos**
  - A veces esto lleva a errores cuando se dan condiciones especiales
- **iCloud: El caso de "Rachel True"**
  - Alguien olvidó forzar el tipo del `lastName` a `string`
  - Por lo tanto, se infirió que era `boolean`
    - No se añadía '' al valor por lo que…
  - El programa falló en ejecución en una capa interna
  - **Consecuencia**: La usuaria fue incapaz de acceder a iCloud durante más de 6 meses
    - Su único "pecado" es tener como apellido "True" …
- **El que puso "null" en la matrícula del coche**
  - Apareció incluso en las noticias
  - ¡Pagó 12k en multas de aparcamiento de otros!



Source: https://blog.elhacker.net/2021/03/usuaria-con-apellido-TRUE-bloquea-sistema-iCloud-Apple-boolean.html

Source: https://www.wired.com/story/null-license-plate-landed-one-hacker-ticket-hell/

Validación de texto Unicode de formato libre

- **El texto de formato libre, especialmente Unicode, a menudo es difícil de validar**
  - Tiene un gran conjunto de caracteres
  - En estos casos también es importante codificar el texto de acuerdo con el contexto de la aplicación
- **Si el texto admite legítimamente caracteres potencialmente peligrosos (', <, or >)…**
  - Deben codificarse correctamente a lo largo del ciclo de vida de esos datos
- **Para validar las entradas de texto de forma libre se usa**
  - **Normalización**: Asegurarse de que hay una codificación canónica que se usa en toda la aplicación
    - No hay caracteres no válidos presentes en ningún momento
  - **Categorías de caracteres permitidos**: Unicode tiene categorías como "dígitos" o "letras"
    - No sólo cubren el alfabeto latino, sino también otros alfabetos globales (Ej.: Árabe, Cirílico...)
  - **Listas blancas de caracteres individuales**: Ej.: Permitir letras y apóstrofes (nombres irlandeses)
    - Pero no toda la categoría de signos de puntuación para un dato determinado
- **Esto está fuera del alcance de la asignatura, pero puedes aprender más aquí:**
  - https://ipsec.Pl/python/2017/input-validation-free-form-unicode-text-python.html

EXPRESIONES REGULARES

- **Las expresiones regulares de AMD o DLP son una herramienta de validación de datos común**
  - Crear expresiones regulares es complejo
  - Hay muchos recursos que **nos proporcionan expresiones regulares ya creadas**
    - **Apropiadas para validar tipos de datos comunes**
    - Uno de los más conocidos es: https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
    - Algunos frameworks también pueden definir expresiones regulares predeterminadas para ciertos tipos de datos
- **El formato PCRE (Perl-Compatible Regular Expression) es un formato muy usado**
  - Muchos lenguajes tienen bibliotecas que procesan expresiones regulares

```
<regex>
  <name>password</name>
  <pattern><![CDATA[^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{4,8}$]]></pattern>
  <description>4 to 8 character password requiring numbers and both lowercase and uppercas
</regex>

<regex>
  <name>complexpassword</name>
  <pattern><![CDATA[^(?:(?=.*\d)(?=.*[A-Z])(?=.*[a-z])|(?=.*\d)(?=.*[^A-Za-z0-9])(?=.*[a-z
=.*[A-Z])(?=.*[a-z])|(?=.*\d)(?=.*[A-Z])(?=.*[^A-Za-z0-9]))(?!.*(.)\1{2,})[A-Za-z0-9!~<>,;:_
\$\^\@\/]{12,128}$]]></pattern>
  <description>12 to 128 character password requiring at least 3 out 4 (uppercase and lowe
and special characters) and no more than 2 equal characters in a row</description>
</regex>

<regex>
  <name>English_digitwords</name>
  <pattern><![CDATA[^(zero|one|two|three|four|five|six|seven|eight|nine)$]]></pattern>
  <description>The English words representing the digits 0 to 9</description>
</regex>

<regex>
  <name>English_daywords</name>
  <pattern><![CDATA[^(Mo|Tu|We|Th|Fr|Sa|Su)$]]></pattern>
  <description>English 2 character abbreviations for the days of the week</description>
</regex>
```

PROTECCIÓN CONTRA ATAQUES DE INYECCIÓN DE CÓDIGO

- **Todos los datos controlados por el usuario recibidos en el servidor deben ser codificados cuando forman parte de la respuesta HTML a una petición**
  - Por ejemplo, `<script>` sería devuelto como `&lt;script&gt;`
- **Si no hacemos esto, facilitamos ataques de inyección de código**
- **El tipo de codificación que necesitan los datos recibidos de los usuarios depende del contexto de la página donde se insertan esos datos**
  - Por ejemplo, la **Codificación de entidades HTML** es apropiada para los datos colocados en el cuerpo HTML de la respuesta
  - Sin embargo, datos de usuario puestos dentro de scripts de una página de respuesta HTTP requerirían **codificación específica de JavaScript**
  - Lo mismo sucede con los datos de usuario colocados dentro de CSS, URLs...
  - **¡La codificación de entidades HTML por sí sola no es suficiente!**
  - Puedes encontrar una guía completa sobre cómo hacerlo aquí (**XSS prevention rules**)
    - https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.md

PROTECCIÓN CONTRA ATAQUES DE INYECCIÓN DE CÓDIGO

- **Aunque hay muchos vectores de ataque XSS, seguir una serie de reglas simples nos permite prevenirlos**
- **Debemos tratar las páginas HTML como una "plantilla" con placeholders**
  - El desarrollador sólo puede colocar datos que no son de confianza dentro de estos placeholders
  - Para cada uno de ellos, implementamos reglas de seguridad ligeramente diferentes
- ***¿Por qué?***
  - Cada placeholder tiene un contexto diferente
  - Las acciones para evitar que el contenido se interprete como código cambian según ese contexto
- **Los datos no fiables nunca deben ser usados en partes de página que no sean esos placeholders definidos en el enlace OWASP de la transparencia anterior**
  - El esfuerzo para asegurarse de que no son un problema es demasiado alto, y no vale la pena
  - Es **necesario** mantener la entrada del usuario dentro de esos placeholders definidos por las reglas
  - **Las reglas 2 y 3 del enlace OWASP anterior cubren la mayoría de los casos comunes**

BIBLIOTECAS DE CODIFICACIÓN

- **No se recomienda codificar manualmente los datos**
  - Hay bibliotecas públicas, gratuitas y muy probadas **que harán un mejor trabajo**
  - Debes codificar tanto **datos de entrada como de salida**
    - Así los datos que produzca nuestra aplicación es menos probable que se usen para atacar a otra
  - La codificación **depende del contexto** en el que se usen los datos
    - Se deben usar métodos distintos si los datos están en el cuerpo de HTML que si están en un parámetro de la URL, por ejemplo
- **Algunos ejemplos son**
  - ASP .NET: https://docs.microsoft.com/es-es/aspnet/core/security/cross-site-scripting?view=aspnetcore-3.1
  - Proyecto OWASP Java Encoder: https://www.owasp.org/index.php/OWASP_Java_Encoder_Project
  - PHP (SEW): https://www.virtuesecurity.com/preventing-cross-site-scripting-php/
  - NodeJS (SDI) usa **motores de plantillas** (template engines) para prevenir XSS con codificación de caracteres
  - Spring Boot (SDI): https://stackabuse.com/securing-spring-boot-web-applications/
    - Debes consultar Spring security para prevenir varios tipos de ataques: https://spring.io/projects/spring-security
  - **OWASP Java Html Sanitizer** cuando la entrada solo acepta un conjunto de etiquetas HTML
    - https://github.com/owasp/java-html-sanitizer

PLANTILLAS DE CODIFICACIÓN AUTOMÁTICA

- **Decidir qué método de codificación usar según el contexto lleva mucho tiempo y es propenso a errores**
- **Muchos frameworks (especialmente los web) tienen plantillas que codifican automáticamente elementos según el contexto**
  - El sistema escoge el método de codificación correcto por nosotros
- **Ejemplos**
  - Strict Contextual Scaping (de AngularJS): https://docs.angularjs.org/api/ng/service/$sce
  - Go Templates (de Go): https://golang.org/pkg/html/template/
  - Motores de plantillas Node.js (para SDI): https://www.veracode.com/blog/secure-development/nodejs-template-engines-whydefault-encoders-are-not-enough
  - Spring Boot (para SDI): Usa las características Spring Security (https://spring.io/guides/gs/securing-web/)
- **Siempre es conveniente ver si nuestro framework tiene este tipo de tecnologías**
  - O elegirlo sólo si posee estas tecnologías, ya que se toma la seguridad en serio

< Ir al Índice

Control de Acceso

Autorización. MFA

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Control de acceso (autorización)

Seguridad de Sistemas Informáticos

- Es mejor tener un mecanismo centralizado de autorización único para toda la aplicación
- Cada petición debe ser autorizada explícitamente
- Restringir el acceso a cualquier cosa valiosa solo para usuarios autorizados
  - Funciones, información...
- Las cuentas de usuario permitidas y sus privilegios deben establecerse con precisión en la fase de diseño

3. Control de acceso (autorización)	
Garantizar el cumplimiento de los principios obligatorios de autorización	
	Los controles de acceso deben fallar de forma segura
	Denegar todo el acceso si la aplicación no puede acceder a su información de configuración de seguridad
	Utilizar un único componente en toda la aplicación para comprobar la autorización de acceso. Esto incluye bibliotecas que llaman a servicios de autorización externos
	Utilizar solo objetos del sistema de confianza (por ejemplo, objetos de sesión del lado del servidor), para tomar decisiones de autorización de acceso
Aplicar las restricciones adecuadas solo a los usuarios autorizados	
	Restringir el acceso a los datos de la aplicación a solo a usuarios autorizados
	Restringir el acceso a archivos u otros recursos, incluidos los que están fuera del control directo de la aplicación, solo a usuarios autorizados
	Restringir el acceso a las funciones protegidas solo a usuarios autorizados
	Restringir el acceso a las URL protegidas solo a usuarios autorizados
	Restringir el acceso a la información de configuración relevante para la seguridad solo a usuarios autorizados
	Restringir el acceso a los servicios solo a usuarios autorizados
	Restringir el acceso a los atributos de los usuarios y datos y a la información de directivas que usan los controles de acceso
	Restringir las referencias directas a objetos solo a usuarios autorizados
Implementar las mejores prácticas de seguridad en el control de acceso en el código de la lógica de negocio	
	Hacer que los flujos de la lógica de la aplicación cumplan obligatoriamente con las reglas de negocio
	Si los datos de estado deben almacenarse en el cliente, utilizar cifrado y la comprobación de integridad en el lado del servidor para detectar la manipulación del estado
	Limitar el número de transacciones que un solo usuario o dispositivo puede realizar en un período de tiempo determinado. Las transacciones / tiempo deben estar por encima del requisito comercial real, pero lo suficientemente bajo como para parar ataques automatizados
	Las representaciones de las reglas de control de acceso en la capa de presentación y en la de implementación del lado del servidor deben coincidir
	Usar la cabecera "referer" solo como verificación suplementaria, nunca como única forma de autorización (se puede falsificar)
Implementar una gestión de autorización adecuada	
	Crear una directiva de control de acceso para documentar las reglas de negocio, los tipos de datos y los criterios y/o procesos de autorización de acceso de una aplicación, de modo que el acceso se pueda crear y controlar correctamente. Esto incluye la identificación de los requisitos de acceso tanto para los datos como para los recursos del sistema
	Aplicar controles de autorización en cada solicitud, incluidas las realizadas por scripts del lado del servidor, "includes" y solicitudes de tecnologías avanzadas del lado del cliente como AJAX
	Si se permiten sesiones autenticadas que duran mucho tiempo, volver a validar periódicamente la autorización de un usuario para asegurarse de que sus privilegios no han cambiado y, si lo han hecho, cerrar su sesión y forzar la reautenticación
	Separar la lógica que requiere privilegios elevados del otro código de aplicación
Administrar cuentas de usuario de forma segura	
	Implementar auditoría de cuentas y forzar la desactivación de las cuentas no utilizadas (por ejemplo, después de 30 días desde la expiración de la contraseña de una cuenta)
	Las cuentas de servicio o las que admiten conexiones hacia o desde sistemas externos deben tener el menor privilegio posible
	La aplicación debe admitir la desactivación de cuentas y la terminación de sesiones cuando cesa la autorización (por ejemplo, cambios en el rol, estado de empleo, proceso comercial, etc.)

4. Autenticación y Gestión de Passwords

Garantizar el cumplimiento de los principios obligatorios de autenticación

- Todos los controles de autenticación deben aplicarse en un sistema de confianza (por ejemplo, el servidor)
- Todos los controles de autenticación deben fallar de forma segura
- Especificar y utilizar servicios de autenticación estándar y probados siempre que sea posible
- Requerir autenticación para todas las páginas y recursos, excepto aquellos destinados a ser públicos
- Separar la lógica de autenticación del recurso que se solicita y utilizar la redirección hacia y desde un control de autenticación centralizado
- Usar una implementación centralizada para todos los controles de autenticación, incluidas las bibliotecas que llaman a servicios de autenticación externos

Implementar políticas de almacenamiento de contraseñas adecuadas

- Las credenciales de autenticación para acceder a servicios externos a la aplicación deben cifrarse y almacenarse en una ubicación protegida en un sistema de confianza (Ej.: el servidor). El código fuente NO es una ubicación segura
- Si la aplicación administra un almacén de credenciales, debe asegurarse de que solo se almacenen hashes de contraseñas salteados unidireccionales criptográficamente fuertes y que la tabla/archivo que almacena las contraseñas y claves solo pueda escribir en ella la aplicación. No usar el algoritmo MD5
- El hash de las contraseñas debe implementarse en un sistema de confianza (por ejemplo, el servidor).

Forzar la transmisión segura de contraseñas

- Solo enviar contraseñas a través de una conexión cifrada o como datos cifrados, como en un correo electrónico cifrado
- Usar solo peticiones HTTP POST para transmitir credenciales de autenticación

Aplicar una directiva de contraseñas seguras

- Hacer cumplir los requisitos de complejidad de contraseñas establecidos por la política o regulación de la aplicación. Las credenciales de autenticación deben ser suficientemente fuertes para resistir los ataques típicos (por ejemplo, forzar el uso de caracteres alfanuméricos y especiales)
- Cambiar todas las contraseñas e IDs de usuario predeterminadas proporcionadas por el proveedor, o deshabilitar las cuentas asociadas
- Aplicar cambios de contraseña en función de los requisitos establecidos en la política de la aplicación. Los sistemas críticos pueden requerir cambios más frecuentes. El tiempo entre reinicios debe controlarse administrativamente
- Hacer cumplir los requisitos de longitud de contraseña establecidos por la política o regulación de la aplicación. Se debe considerar el uso de frases de contraseña de varias palabras
- Las contraseñas deben tener al menos un día de antigüedad antes de que puedan cambiarse, para evitar ataques a la reutilización de contraseñas
- Impedir la reutilización de contraseñas

Implementar prácticas seguras para mecanismos de autenticación

- Validar los datos de autenticación solo al completar todas las entradas de datos, especialmente para implementaciones de autenticación secuencial
- Las respuestas de error de autenticación no deben indicar qué parte de los datos de autenticación era incorrecta. Por ejemplo, en lugar de "Nombre de usuario no válido" o "Contraseña no válida", debe usarse "Nombre de usuario y / o contraseña no válidos" para ambos. Las respuestas de error deben ser verdaderamente idénticas tanto en la pantalla como en el código fuente
- Si se utiliza código de terceros para la autenticación, revisar el código cuidadosamente para asegurarse de que no tenga ningún código malicioso
- Utilice la autenticación multifactor para cuentas de alta sensibilidad o alto valor
- Utilizar la autenticación para conexiones a sistemas externos que involucran información o funciones confidenciales

Administrar las contraseñas de las cuentas de forma segura en el GUI / lógica de la aplicación

- Todas las funciones administrativas y de administración de cuentas deben ser tan seguras como el mecanismo de autenticación principal
- Deshabilitar la funcionalidad "recordar" para los campos de contraseña
- Desactivar una cuenta después de un número preestablecido de intentos de inicio de sesión no válidos (por ejemplo, es común cinco intentos). La cuenta debe deshabilitarse durante un período de tiempo suficiente para impedir adivinar credenciales por fuerza bruta, pero no tanto como para permitir que se realice un ataque de denegación de servicio
- Aplicar el cambio de contraseñas temporales en el próximo uso de la cuenta
- Si se usa el restablecimiento de contraseña por correo electrónico, solo enviar el correo a una dirección preregistrada, con un enlace / contraseña temporal
- Notificar a los usuarios cuando se produce un restablecimiento de contraseña
- La entrada de la contraseña debe estar oculta en la pantalla del usuario (por ejemplo, en los formularios web utilizar el tipo de entrada "contraseña")
- Las operaciones de restablecimiento y cambio de contraseña deben tener el mismo nivel de control que la creación y autenticación de cuentas
- Las preguntas de restablecimiento de contraseña deben admitir respuestas suficientemente aleatorias. (por ejemplo, "libro favorito" es una mala pregunta porque "La Biblia" es una respuesta muy común). No obstante, es mejor no usar este mecanismo de restablecimiento
- Volver a autenticar a los usuarios antes de realizar operaciones críticas
- Las contraseñas y enlaces temporales deben tener un tiempo de caducidad corto

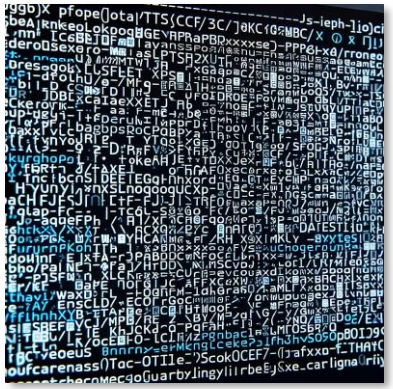
Implementar un log de contraseñas

- Implementar monitorización para identificar ataques contra múltiples cuentas de usuario utilizando la misma contraseña (password spray). Este patrón de ataque se utiliza para eludir los bloqueos estándar, cuando los ID de usuario se pueden recolectar o adivinar
- Debe informarse al usuario del último uso (exitoso o fallido) de una cuenta de usuario en su próximo inicio de sesión

Implementación de un MFA

- **La implementación de un MFA puede parecer algo muy avanzado y difícil**
  - Usar MFA es tan común que hay APIs que simplifican mucho su implementación
- **Debemos buscar la mejor API para nuestro framework / lenguaje**
- **Y, si es posible, utilizar soluciones de MFA de terceros existentes**
- **Algunos ejemplos de implementación**
  - Google OAuth2: https://developers.google.com/identity/protocols/OAuth2
  - Google Authenticator: https://medium.com/@richb_/easy-two-factor-authentication-2fa-with-googleauthenticator-php-108388a1ea23
  - PHP (para SEW): https://medium.com/@richb_/easy-two-factor-authentication-2fa-with-googleauthenticator-php-108388a1ea23
  - Node .js (para SDI): https://developer.okta.com/blog/2018/05/22/simple-multifactor-authentication-innode
  - Spring Boot (para SDI): https://sultanov.dev/blog/multi-factor-authentication-with-spring-boot-andoauth2/

< Ir al Índice



🤫 CRIPTOGRAFÍA Y SEGURIDAD DE LOS DATOS

Como proteger tus datos privados

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Consejos prácticos sobre criptografía

- No expongas información secreta
- **Usa un vault** para guardarla
  - Tu framework debería tener uno
  - Hay también soluciones de terceros
- Cifra la información privada en las bases de datos
- No uses módulos criptográficos no fiables, obsoletos o no validados
  - Sigue los principios segundos en el **Tema 2** y **6**

5. Prácticas criptográficas	
Garantizar el cumplimiento obligatorio de las políticas de gestión de la criptografía	
	Todas las funciones criptográficas utilizadas para proteger los secretos del usuario de la aplicación deben implementarse en un sistema de confianza (por ejemplo, el servidor)
	Establecer y utilizar una política y un proceso sobre cómo se administrarán las claves criptográficas
	Proteger las claves o secretos maestros de accesos no autorizados
Usa solo módulos de criptografía adecuados	
	Todos los números aleatorios, nombres de archivos aleatorios, GUID aleatorios y cadenas aleatorias deben generarse utilizando un generador de números aleatorios recomendado por el módulo utilizado criptográfico cuando estos valores aleatorios están destinados a no poder ser predichos
	Los módulos criptográficos deben fallar de forma segura
	Los módulos criptográficos utilizados por la aplicación deben cumplir con FIPS 140-2 o un estándar equivalente. (Ver http://csrc.nist.gov/groups/STM/cmvp/validation.html)

SEGURIDAD DE LAS COMUNICACIONES

- **Para evitar problemas y garantizar la máxima protección, siempre usar TLS**
  - Sin importar el contenido de los datos transmitidos
  - **¡HTTPS siempre es obligatorio en producción!**
- **Usa siempre una versión de TLS segura**
- **Usa una configuración adecuada para TLS en el servidor web**
  - Esto no es difícil si seguimos las recomendaciones de Mozilla
  - https://ssl-config.mozilla.org/

6. Seguridad de las comunicaciones	
Garantizar el cumplimiento de los principios obligatorios de seguridad en las comunicaciones	
	Filtrar parámetros que contienen información confidencial del referente HTTP cuando se enlace a sitios externos
	Implementar cifrado para la transmisión de toda la información confidencial (o para toda la información, en general). Esto debe incluir TLS para proteger la conexión y puede complementarse con cifrado discreto de archivos confidenciales o conexiones no basadas en HTTP
	Especificar codificaciones de caracteres para todas las conexiones
Implementar directivas de administración de TLS seguras	
	Las conexiones TLS fallidas no deben intentar usar una conexión insegura como mecanismo de fallo
	Los certificados TLS deben ser válidos y tener el nombre de dominio correcto, no caducar e instalarse con certificados intermedios cuando sea necesario
	Utilizar una única implementación estándar de TLS que esté configurada adecuadamente
	Utilizar conexiones TLS para todo el contenido que requiera acceso autenticado y para cualquier otra información confidencial
	Utilizar TLS para conexiones a sistemas externos que involucran información o funciones confidenciales

< Ir al Índice

😍⌨️ PRÁCTICAS GENERALES PARA UN CÓDIGO SEGURO

Acciones concretas a la hora de crear vuestro código

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

PRÁCTICAS GENERALES DE CREACIÓN DE CÓDIGO

Seguridad de Sistemas Informáticos

- **Actualizar todo lo que se use**
  - Una vulnerabilidad (y su CVE) pueden ser la raíz de un problema
- **Usar medidas para evitar falsificación de dependencias de terceros**
  - Hashes (**Tema 2**) o herramientas SCA (**Tema 6**)
- **No uses metaprogramación**
  - TPP: `eval`, `exec`...
- **Exponerse el tiempo mínimo posible**
  - Minimizar el acceso con privilegios
  - Implementa medidas para impedir el reversing (**Seminario 4**), como ofuscación de código
- **Programar defensivamente**
  - https://latteandcode.medium.com/qu%c3%a9-es-eso-de-la-programaci%c3%b3n-defensiva6a89d3bcab8e

7. Prácticas generales de creación de código

Garantizar el cumplimiento de las prácticas obligatorias de codificación segura	
	Usar código administrado probado y aprobado en lugar de crear nuevo código no administrado para tareas comunes
Implementar una política segura para usar API / Bibliotecas externas	
	Implementar actualizaciones seguras. Si la aplicación va a usar actualizaciones automáticas, usar firmas criptográficas para su código y asegurarse de que los clientes de descarga verifican esas firmas. Usar canales cifrados para transferir el código desde el servidor host
	Revisar todas las aplicaciones secundarias, el código de terceros y las bibliotecas para determinar si son necesarias y validar que su funcionalidad es segura, ya que pueden introducir nuevas vulnerabilidades
	Usar sumas de comprobación o hashes para comprobar la integridad del código interpretado, las bibliotecas, los ejecutables y los archivos de configuración
	Utilizar API integradas específicas para realizar tareas del sistema operativo. No permitir que la aplicación emita comandos directamente al sistema operativo, especialmente mediante shells de comandos iniciados por la aplicación
Utilizar técnicas de programación segura	
	Evitar errores de cálculo comprendiendo la representación subyacente del lenguaje de programación y cómo interactúa con el cálculo numérico. Prestar mucha atención a las discrepancias de tamaño de bytes, la precisión, las distinciones con signo / sin signo, el truncamiento, la conversión y el cast entre tipos, los cálculos de "no un número" y la forma en que el lenguaje maneja los números que son demasiado grandes o demasiado pequeños para su representación subyacente
	No pasar los datos proporcionados por el usuario a ninguna función de ejecución dinámica
	Inicializar explícitamente todas sus variables y otros almacenes de datos, ya sea durante la declaración o justo antes del primer uso
	En los casos en que la aplicación deba ejecutarse con privilegios elevados, incrementar los privilegios lo más tarde posible y volver al nivel de privilegio normal lo antes posible
	Proteger las variables y los recursos compartidos del acceso simultáneo inapropiado
	Impedir que los usuarios generen código nuevo o alteren el código existente
	Utilizar bloqueos para evitar múltiples solicitudes simultáneas o utilizar un mecanismo de sincronización para evitar las condiciones de carrera
(EXTRA) Características contra la ingeniería inversa	
	Usa un packer
	[illegible]

PROTECCIÓN DE DATOS

Seguridad de Sistemas Informáticos

- **Aplicar siempre al acceso a los datos el menor privilegio necesario**
- **No dar información sobre uno mismo**
  - Usuarios, personal, desarrolladores…, la estructura de la aplicación o las tecnologías que se usan
- **Evitar recordar datos confidenciales en la interfaz gráfica de la aplicación**
- **Proteger el código fuente a toda costa**
- **Estas técnicas se trataron en**
  - **Tema 2** (criptografía)
  - **Tema 3** (seguridad del SO)
  - **Tema 6** (seguridad de aplicaciones)

8. Protección de datos	
Implementar el manejo seguro de permisos para datos	
	Implementar controles de acceso adecuados para los datos confidenciales almacenados en el servidor. Esto incluye datos almacenados en caché, archivos temporales y datos a los que solo deben acceder usuarios específicos del sistema
	Implementar privilegios mínimos, restringir a los usuarios solo a la funcionalidad, los datos y la información del sistema que sean estrictamente necesarios para realizar sus tareas
	Proteger todas las copias en caché o temporales de los datos confidenciales almacenados en el servidor del acceso no autorizado y purgar esos archivos temporales tan pronto como ya no sean necesarios
	Proteger el código fuente del lado del servidor para que un usuario no pueda descargarlo
Cifrado de datos de forma segura	
	Cifrar la información almacenada altamente confidencial, como los datos de verificación de autenticación, incluso en el lado del servidor. Utilizar siempre algoritmos considerados adecuados y consultar "Prácticas criptográficas" para obtener orientación adicional
Implementar almacenamiento seguro de información	
	No almacenar contraseñas, cadenas de conexión u otra información confidencial en texto sin cifrar o de cualquier manera no criptográficamente segura en el lado del cliente. Esto incluye la incrustación de información en formatos inseguros como: MS viewstate o código compilado
	La aplicación debe admitir la eliminación de datos confidenciales cuando esos datos ya no sean necesarios. (por ejemplo, información personal o ciertos datos financieros)
Eliminar cualquier información que la aplicación proporcione sobre sí misma	
	No incluir información confidencial en los parámetros de solicitud HTTP GET
	Eliminar comentarios en el código en producción accesible por el usuario que pueda revelar datos del backend u otra información confidencial
	Eliminar metadatos de archivos y cualquier tipo de información adicional que los archivos puedan dar
	No usar plantillas predeterminadas de estructuras que puedan revelar tecnologías que esté utilizando
	Eliminar la documentación innecesaria de la aplicación y el sistema, ya que esto puede revelar información útil a los atacantes
Implementar prácticas de GUI seguras para tratar con datos confidenciales	
	Deshabilitar las funciones de autocompletado en formularios que se espera que contengan información confidencial, incluida la autenticación
	Deshabilitar el almacenamiento en caché del lado del cliente en páginas que contengan información confidencial. Cache-Control: no-store, se puede usar junto con el control de la cabecera HTTP "Pragma: no-cache", que es menos efectivo, pero es compatible con versiones anteriores de HTTP / 1.0

GESTIÓN DE SESIONES

Seguridad de Sistemas Informáticos

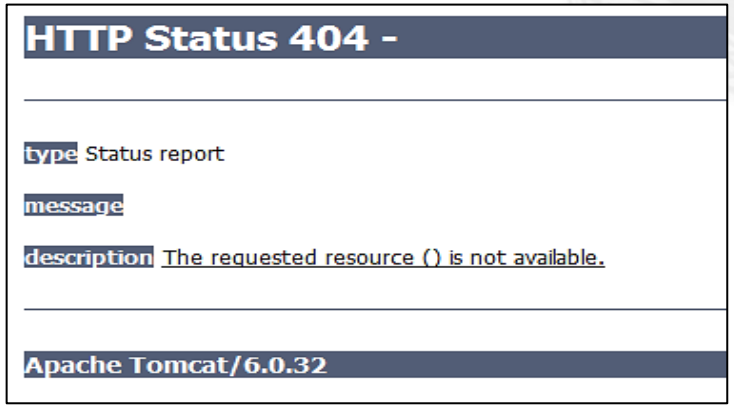
- **Robar una sesión significa hacerse pasar por un usuario: DEBE evitarse**
  - Nunca revelar un ID de sesión válido
  - Sigue las reglas de manejo seguro de IDs de sesión recomendadas
- **Seguir las pautas para implementar cookies de forma segura**
  - Los ID de sesión normalmente se almacenan como atributos de cookies
- **Controlar comportamientos sospechosos de los usuarios**
  - Sesiones simultáneas desde diferentes geolocalizaciones...
- **Asegurarse de que el cierre de sesión realmente la invalida**

9. Gestión de sesiones	
Garantizar el cumplimiento de las prácticas obligatorias de gestión de sesiones	
	La creación del identificador de sesión siempre debe realizarse en un sistema de confianza (por ejemplo, el servidor)
	Los controles de gestión de sesiones deben utilizar algoritmos validados que garanticen identificadores de sesión suficientemente aleatorios
	Utilizar los controles de administración de sesiones del servidor o del framework. La aplicación solo debe reconocer estos identificadores de sesión como válidos
Implementar una gestión segura de cookies	
	No exponer identificadores de sesión en URLs, mensajes de error o logs. Los identificadores de sesión solo deben estar en la cabecera HTTP de la cookie. Por ejemplo, no pasar identificadores de sesión como parámetros GET
	Establecer cookies con el atributo HttpOnly, a menos que se necesiten específicamente scripts del lado del cliente dentro de la aplicación para leer o establecer el valor de una cookie
	Establecer el atributo "secure" para las cookies transmitidas a través de una conexión TLS
	Establecer el dominio y la ruta de acceso a un valor restringido adecuadamente para el sitio para las cookies que contienen identificadores de sesión autenticados
Implementar una gestión segura de inicios de sesión	
	No permitir inicios de sesión simultáneos con el mismo ID de usuario, salvo que sea un requisito
	Generar un nuevo identificador de sesión en cualquier reautenticación
	Si se estableció una sesión antes de iniciar sesión, cerrar esa sesión y establecer una nueva después de un inicio de sesión exitoso
Implementar una administración segura de cierres de sesión	
	No permitir los inicios de sesión persistentes y forzar la terminación periódica de la sesión, incluso cuando la sesión esté activa. Especialmente para aplicaciones que se conecten a sistemas críticos. Los tiempos de terminación deben respaldar los requisitos del negocio y el usuario debe recibir una notificación con antelación suficiente para mitigar los efectos negativos
	Establezca un tiempo de espera de inactividad de sesión que sea lo más corto posible, estableciendo un equilibrio entre riesgo y los requisitos funcionales del negocio. En la mayoría de los casos no debe ser más de unas pocas horas
	La funcionalidad de cierre de sesión debe estar disponible en todas las páginas protegidas por autorización
	La funcionalidad de cierre de sesión debe finalizar completamente la sesión o conexión asociada
Manejar los datos de sesión de forma segura	
	Generar un nuevo identificador de sesión y desactivar el anterior periódicamente. (Esto puede mitigar ciertos escenarios de secuestro de sesiones en los que el identificador original se vio comprometido)
	Generar un nuevo identificador de sesión si la seguridad de la conexión cambia de HTTP a HTTPS, como puede ocurrir durante la autenticación. Dentro de una aplicación, se recomienda utilizar HTTPS de manera consistente en lugar de cambiar entre HTTP a HTTPS
	Proteger los datos de sesión del lado del servidor del acceso no autorizado por parte de otros usuarios del servidor mediante la implementación de controles de acceso adecuados en el servidor
	Complementar la administración de sesiones estándar para operaciones altamente sensibles o críticas utilizando tokens o parámetros aleatorios fuertes por cada petición, en lugar de por cada sesión
	Complementar la administración de sesiones estándar para operaciones confidenciales del lado del servidor, como la administración de cuentas, utilizando tokens o parámetros aleatorios fuertes por sesión. Este método se puede utilizar para prevenir ataques de falsificación de solicitudes entre sitios (CSRF)

MANEJO/GESTIÓN DE ERRORES

Seguridad de Sistemas Informáticos

- **Los errores pueden mostrar mucha información**
  - Mensajes genéricos para el usuario, detallados para desarrolladores
  - Evitar errores como el de la imagen
- **Haz log de los errores importantes**
  - ¡Asegúrate de que no revelen información privada!
- **Limita el acceso a los logs**
  - Únicamente para usuarios autorizados



10. Manejo y log de errores	
Garantizar el cumplimiento de las prácticas de log obligatorias	
	Todos los controles de log deben implementarse en un sistema de confianza (por ejemplo, el servidor)
No divulgar información excesiva sobre los mensajes de error	
	No divulgar información confidencial en las respuestas de error, incluidos los detalles del sistema, los identificadores de sesión o la información de la cuenta
	Implementar mensajes de error genéricos y usar páginas de error personalizadas
	Usar controladores de errores que no muestren información de depuración o de la pila de llamadas
Implementar el manejo seguro de errores	
	La lógica de control de errores asociada con los controles de seguridad debería denegar el acceso de forma predeterminada
	Liberar correctamente la memoria asignada cuando se producen condiciones de error
	La aplicación debe controlar los errores de la aplicación y no depender de la configuración del servidor
Habilitar la creación y administración segura de log de errores	
	No almacenar información confidencial en logs, incluidos detalles innecesarios del sistema, identificadores de sesión o contraseñas
	Asegurarse de que las entradas de log que incluyen datos no confiables no se ejecutarán como código en la interfaz de visualización de logs o el software previsto para ello
	Asegurarse de que los logs contengan datos importantes de los eventos registrados
	Asegurarse de que existe un mecanismo para realizar análisis de logs
	Los controles de log deben admitir tanto el éxito como el fracaso de los eventos de seguridad especificados
	Restringir el acceso a los logs solo a personas autorizadas
	Usar una función hash criptográfica para validar la integridad de las entradas de log
	Usar una función "maestra" única para todas las operaciones de registro
Registrar los tipos adecuados de información relacionada con la seguridad	
	Registrar todos los errores de control de acceso
	Registrar todas las funciones administrativas, incluidos los cambios en las opciones de configuración de seguridad
	Registrar todos los eventos de manipulación aparentes, incluidos cambios inesperados en los datos de estado
	Registrar todos los intentos de autenticación, especialmente los errores
	Registrar todos los errores de conexión TLS al back-end
	Registrar todos los errores de validación de entrada
	Registrar intentos de conexión con tokens de sesión no válidos o caducados
	Registrar todas las excepciones del sistema
	Registrar errores de módulos criptográficos

GESTIÓN DE ARCHIVOS

Seguridad de Sistemas Informáticos

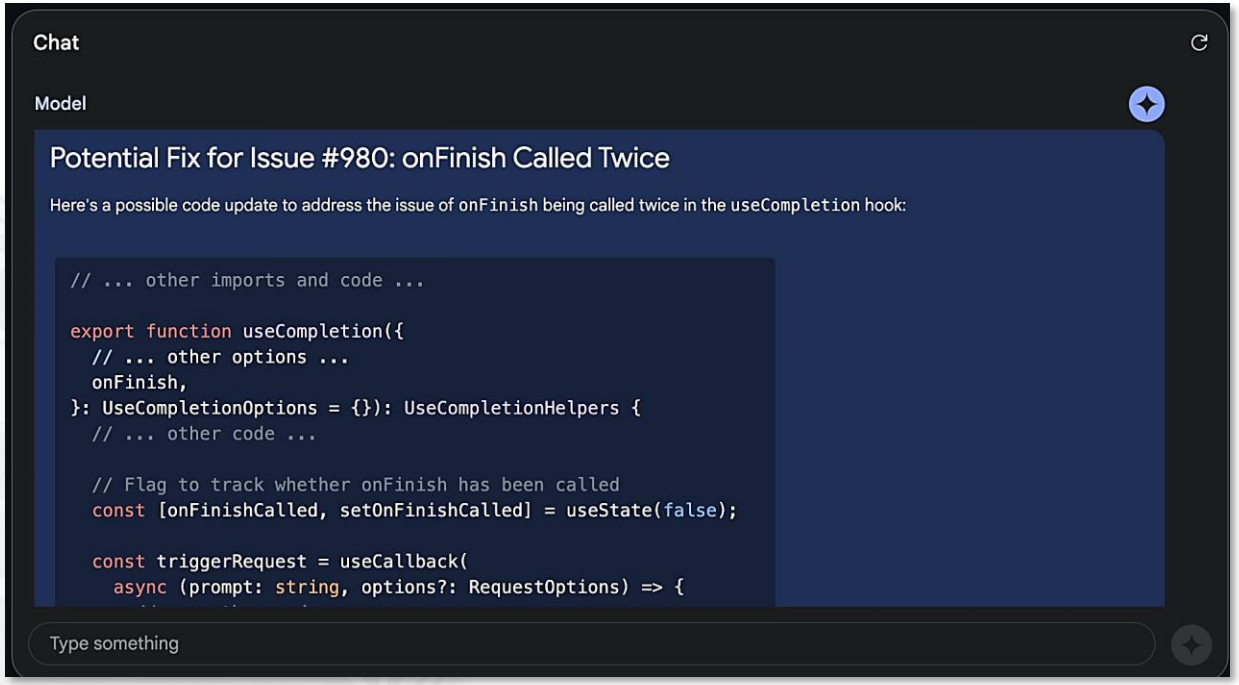
- **Trata de evitar tener una funcionalidad de carga de archivos en la aplicación**
- **Si debe hacerse, sigue las reglas de carga de archivos recomendadas**
  - https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html
  - Especialmente las que previenen las subidas de archivos maliciosos
  - O, mejor, **usar un módulo de terceros probado y verificado que lo implemente**
- **Y recuerda...**
  - **Comprobar el acceso a archivos externos**, el usuario podría indicar uno inválido
  - **No permitir que los usuarios especifiquen rutas**
    - ¡Incluso parcialmente!
  - **Usa listas** de tipos de archivos permitidos

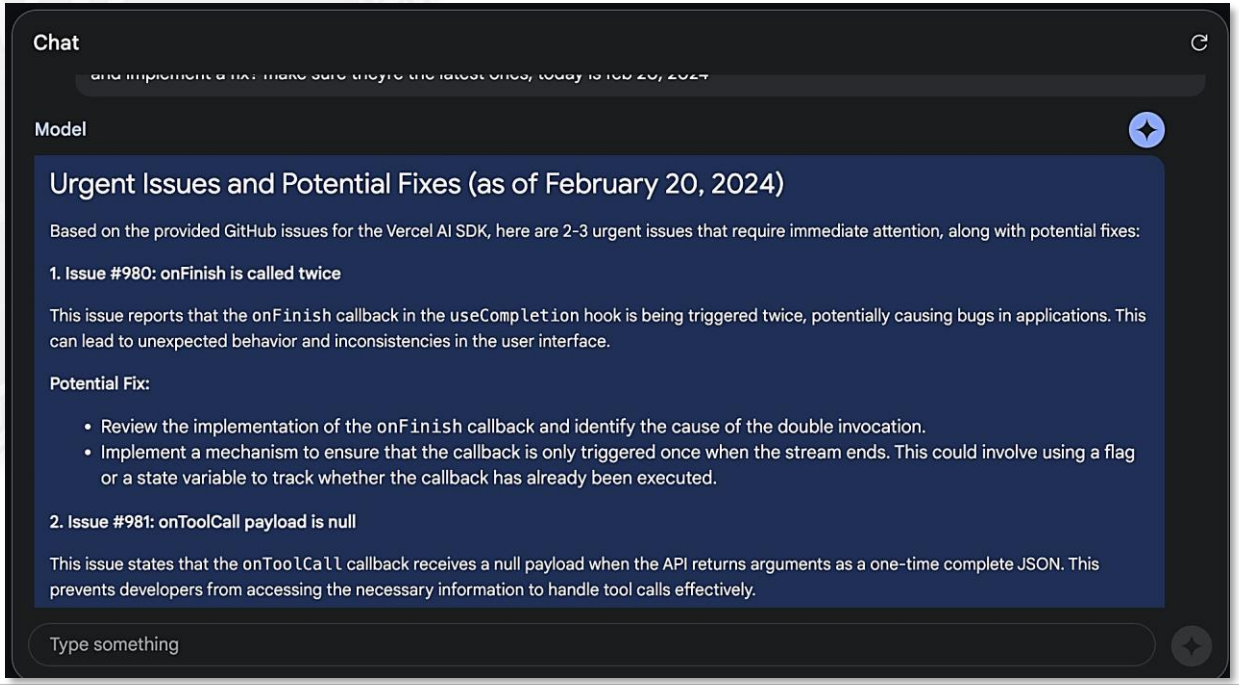
11. Manejo de ficheros	
Garantizar el cumplimiento de las prácticas obligatorias de administración de archivos	
	Asegurarse de que los archivos y recursos de la aplicación sean de solo lectura
Manejar de forma segura las inclusiones y referencias a los archivos	
	No pasar los datos proporcionados por el usuario directamente a ninguna función de inclusión dinámica
	No pasar rutas de directorio o archivo, usar índices asignados a una lista predefinida de rutas de acceso
	No pasar los datos proporcionados por el usuario a una redirección dinámica. Si esto debe permitirse, la redirección debe aceptar solo URLs de rutas de acceso relativas validadas
	Nunca enviar la ruta absoluta de un archivo al cliente
	Al hacer referencia a archivos existentes, utilizar una lista blanca de nombres y tipos de archivo permitidos. Validar el valor del parámetro que se está pasando y, si no coincide con uno de los valores esperados, rechazarlo o usar un archivo predeterminado para el contenido en su lugar
Implementar cargas de archivos seguras, si esta funcionalidad es necesaria	
	No guardar archivos en el mismo contexto web que la aplicación. Los archivos deben ir al servidor de contenido o a la base de datos
	Implementar la carga segura en Linux montando el directorio de archivos de destino como una unidad lógica utilizando la ruta asociada o un entorno chrooted
	Limitar el tipo de archivos que se pueden cargar solo a los tipos que se necesiten para la aplicación
	Impedir o restringir la carga de cualquier archivo que pueda ser interpretado por el servidor web
	Requerir autenticación antes de permitir que se cargue un archivo
	Analizar los archivos cargados por el usuario en busca de virus y malware
	Desactivar los privilegios de ejecución en los directorios de carga de archivos
	Validar los archivos cargados son del tipo esperado comprobando los encabezados de archivo. La comprobación del tipo de archivo sólo por su extensión no es suficiente

SAST ASISTIDO POR IA

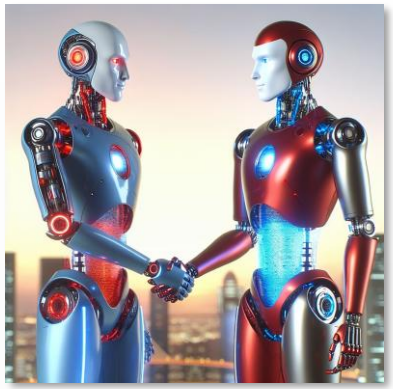
Seguridad de Sistemas Informáticos

- **Estudia si usar IA en tu beneficio para desarrollar código más seguro**
  - Validación de datos, documentación... y **detección de vulnerabilidades de código**
- **Por ejemplo, este usuario usa una IA como herramienta SAST avanzada**
  - https://twitter.com/SullyOmarr/status/1760066335898513655
  - Es Gemini 1.5 de Google
  - Cargó un repositorio de código completo directamente desde GitHub, y todos sus issues
  - Fue capaz de **entender todo el código**
    - Identificó el problema más urgente
    - ¡E implementó una solución!





< Ir al Índice



➡️ Configuración del sistema y dependencias

Seguridad en la configuración de la aplicación

Departamento de Informática
Universidad de Oviedo
Universidad de Oviedo

CONFIGURACIÓN DEL SISTEMA

Seguridad de Sistemas Informáticos

- **En el servidor de producción...**
  - No cargar contenidos innecesarios
  - No habilitar funcionalidades innecesarias
  - Configurarlo de forma segura
    - Siguiendo procedimientos validados de hardening automatizado (tema 4)
    - CIS Benchmarks, STIGs, OpenSCAP...
- **Si es un servidor web, la instalación básica de Apache / IIS se ve en ASR**

13. Configuración del sistema	
Usar una directiva de actualización de software segura	
	Asegurarse de que los servidores, frameworks y componentes del sistema ejecutan la última versión aprobada
	Asegurarse de que los servidores, frameworks y componentes del sistema tengan todos los parches emitidos para la versión en uso
Implementar una configuración de servidor web segura con respecto a los elementos de software	
	Definir qué métodos HTTP, GET o POST, admitirá la aplicación y si se manejarán de manera diferente en diferentes páginas de la aplicación
	Deshabilitar los métodos HTTP innecesarios, como las extensiones WebDAV. Si se requiere un método HTTP extendido que admita el manejo de archivos, utilizar un mecanismo de autenticación validado
	Si el servidor web maneja HTTP 1.0 y 1.1, asegurarse de que ambos estén configurados de una forma similar o asegurarse de comprender cualquier diferencia que pueda existir (por ejemplo, el manejo de métodos HTTP extendidos)
	Eliminar la información innecesaria de los encabezados de respuesta HTTP relacionados con el sistema operativo, la versión del servidor web y los frameworks de aplicaciones
	Restringir las cuentas del servidor web, de su proceso y del servicio al menor número de privilegios posibles
	Desactivar listados de directorios
Minimizar y eliminar archivos innecesarios en el servidor	
	Evitar la divulgación de la estructura de directorios en el archivo robots.txt colocando directorios no destinados a la indexación pública en un directorio principal aislado. Luego, "Disallow" todo ese directorio principal en el archivo robots.txt en lugar de no permitir cada directorio individual
	Eliminar toda la funcionalidad y los archivos innecesarios
	Quitar el código de prueba o cualquier funcionalidad no destinada a producción antes del despliegue
Implementar una directiva de configuración de aplicaciones segura	
	Implementar un sistema de control de cambios de software para gestionar y registrar los cambios en el código, tanto en desarrollo como en producción
	Implementar un sistema de gestión de activos y registrar los componentes y el software del sistema en él
	Aislar los entornos de desarrollo de la red de producción y proporcionar acceso solo a grupos de desarrollo y prueba autorizados. Los entornos de desarrollo a menudo se configuran de forma menos segura que los entornos de producción, y los atacantes pueden utilizar esta diferencia para descubrir debilidades compartidas o como una vía de explotación.
	El almacén de configuración de seguridad de la aplicación debe poder generarse de forma legible por humanos para admitir la auditoría
	Cuando se producen excepciones, fallar de forma segura

SEGURIDAD DE LA BASE DE DATOS

Seguridad de Sistemas Informáticos

- **USAR SIEMPRE CONSULTAS PARAMETRIZADAS**
- **Tratar la base de datos como cualquier sistema externo**
  - Codificar la entrada adecuadamente
  - No confiar en los datos que devuelve; validarlos
  - Actualizar el software del SGBD
- **Usar una política de administración de seguridad**
  - Contraseñas (fuertes, no predeterminadas...)
  - Eliminar contenidos innecesarios
  - Y usar el correspondiente CIS Benchmark (o similar) para la configuración segura del SGBD

14. Seguridad de la base de datos	
Garantizar el cumplimiento de los principios obligatorios de administración segura de bases de datos	
	Usar consultas parametrizadas fuertemente tipadas
Manejo seguro de tipos	
	Asegurarse de que las variables estén fuertemente tipadas
	Utilizar la validación de entrada y la codificación de salida y asegúrese de manejar los meta caracteres. Si estos fallan, no ejecutar el comando de la base de datos asociado
Aplicar el permiso de base de datos segura adecuado	
	La aplicación debe conectarse a la base de datos con diferentes credenciales para cada nivel de confianza (por ejemplo, usuario, usuario de solo lectura, invitado, administradores).
	La aplicación debe utilizar el nivel de privilegios más bajo posible al acceder a la base de datos
Implementar políticas seguras de administración de bases de datos	
	Cerrar la conexión lo antes posible
	Las cadenas de conexión no deben estar codificadas dentro de la aplicación. Las cadenas de conexión deben almacenarse en un archivo de configuración independiente en un sistema de confianza y deben cifrarse
	Deshabilitar las cuentas predeterminadas que no sean necesarias para los requisitos de negocio
	Quitar o cambiar todas las contraseñas administrativas predeterminadas de la base de datos. Utilizar contraseñas/frases seguras o implementar autenticación multifactor
	Eliminar el contenido del proveedor predeterminado innecesario (por ejemplo, esquemas de ejemplo)
	Desactivar toda la funcionalidad innecesaria de la base de datos (por ejemplo, procedimientos o servicios almacenados innecesarios, paquetes de utilidades, instalare solo un conjunto mínimo de características y opciones requeridas (reducción del área de ataque))
	Usar credenciales seguras para el acceso a la base de datos
	Usar procedimientos almacenados para abstraer el acceso a los datos y permitir la eliminación de permisos para las tablas base de la base de datos

Uso seguro de frameworks / lenguajes de desarrollo

- **Minimice las bibliotecas / dependencias del marco tanto como sea posible**
  - Cada dependencia es una vulnerabilidad potencial presente o futura
  - **El caso `xz`**: Un desarrollador deshonesto de esta biblioteca comprometió el servicio SSH de todos los servidores que lo usaban para ejecutar comandos arbitrarios
    - ¡Fue descubierto casualmente! (por lo tanto, cuantas menos dependencias, menos riesgo)
- **Los frameworks / lenguajes modernos tienen protecciones anti-XSS**
  - Deben usarse correctamente en nuestras aplicaciones (**nunca validar manualmente**)
  - **No las desactives**
- **Utilizar siempre la última versión estable posible (parches)**
  - Desplegar en producción usando las opciones del framework (Ej.: Angular 2+: `ng build-prod`)
- **Muchos frameworks/lenguajes integran protecciones de seguridad**
  - Usa todas las que se apliquen a tu aplicación
- **Puedes encontrar cheatsheets acerca de desarrollo seguro en:**
  - https://cheatsheetseries.owasp.org/Glossary.html

Lista de logros para autoevaluación: Seminario 5

?

- **Eres capaz de entender lo siguiente**
  - *La diferencia y la aplicación de una lista blanca y una lista negra*
  - *Cómo hacer conversiones de tipo y comprobaciones de rango apropiadas*
  - *Cómo usar bibliotecas de encoding*
  - *Cómo usar bibliotecas de codificación y "saneadores" HTML adecuados*
  - *Cómo elegir los requisitos de seguridad más adecuados*
  - *Las estrategias de validación adaptadas a cada contexto*
  - *Los mecanismos de validación de los frameworks web modernos*
  - *Todos los aspectos generales para crear un código seguro*
  - *Cómo tratar la funcionalidad de subir archivos a una aplicación*
  - *Las reglas para prevenir ataques XSS*

😟 SEMINARIO 5
VALIDACIÓN DE ENTRADAS Y TÉCNICAS DE DESARROLLO SEGURO

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo