

SEMINARIO 5

VALIDACIÓN DE ENTRADAS Y TÉCNICAS PARA CREAR CÓDIGO SEGURO



Estrategias para rechazar datos
inválidos



JOSÉ MANUEL REDONDO LÓPEZ PROYECTO "S-81 ISAAC PERAL" v4.0



Departamento de Informática
Universidad de Oviedo



Universidad de Oviedo

Logo: @creative_vanesa (Instagram)

¿QUÉ ES ESTE SEMINARIO?



Seguridad de Sistemas
Informáticos

- **En este seminario vamos a dar ejemplos prácticos de cómo hacer determinadas 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**)
 - 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
 - Esta pensado para que lo uséis en vuestro TFG o proyectos software futuros



ÍNDICE

- ▶ Requisitos de seguridad
- ▶ Validación de entradas y codificación de salida
- ▶ Control de acceso
- ▶ Criptografía y seguridad de datos
- ▶ Prácticas generales de creación de código seguro
- ▶ Configuración del sistema, herramientas de desarrollo y dependencias

[< 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



● 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

● Una vez tengas soltura con ella debes usar una más completa

- La OWASP* Application Security Verification Standard (ASVS)
 - Gratuita, internacional y validada
 - <https://github.com/OWASP/ASVS/tree/v4.0.3#latest-stable-version---403>
- Usable también en las siguientes fases: **diseño, implementación, y testing**

● 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



Seguridad de Sistemas
Informáticos

● 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
- RS8.5: 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



- **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
- **En cualquier caso, la validación de la entrada debe**
 - Aplicarse a todos los datos de entrada
 - Definir un conjunto permitido de caracteres aceptados
 - Definir una longitud mínima y máxima
 - Incluidos dígitos decimales y no decimales, el tamaño máximo de los nombres...
 - Definir un tipo o rango esperado si es posible

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 una sintaxis y estructura correctas: Fechas, moneda, DNIs, tfnos...
 - **Semántica** obliga a los valores a ser correctos en un contexto o de acuerdo con las reglas de negocio dadas
 - 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
 - Debe hacerse de forma planificada y exhaustiva, como parte de los requisitos del sistema
 - Si es un caso claro de manipulación maliciosa, se deben **implementar respuestas contra el usuario (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 conocidos** 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**: El número de nodos de salida en la red ToR en un momento determinado 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...) es factible definir un patrón de validación muy fuerte, **generalmente con expresiones 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
 - **Conversión de tipos** (Ej. `Integer.parseInt()` en Java, `int()` en Python...) y **control estricto de excepciones**
 - **Comprobación del valor mínimo y máximo** para los 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 (ver el seminario de validación de entradas)
 - Se debe validar toda la cadena de entrada, **evitando el uso de comodines**

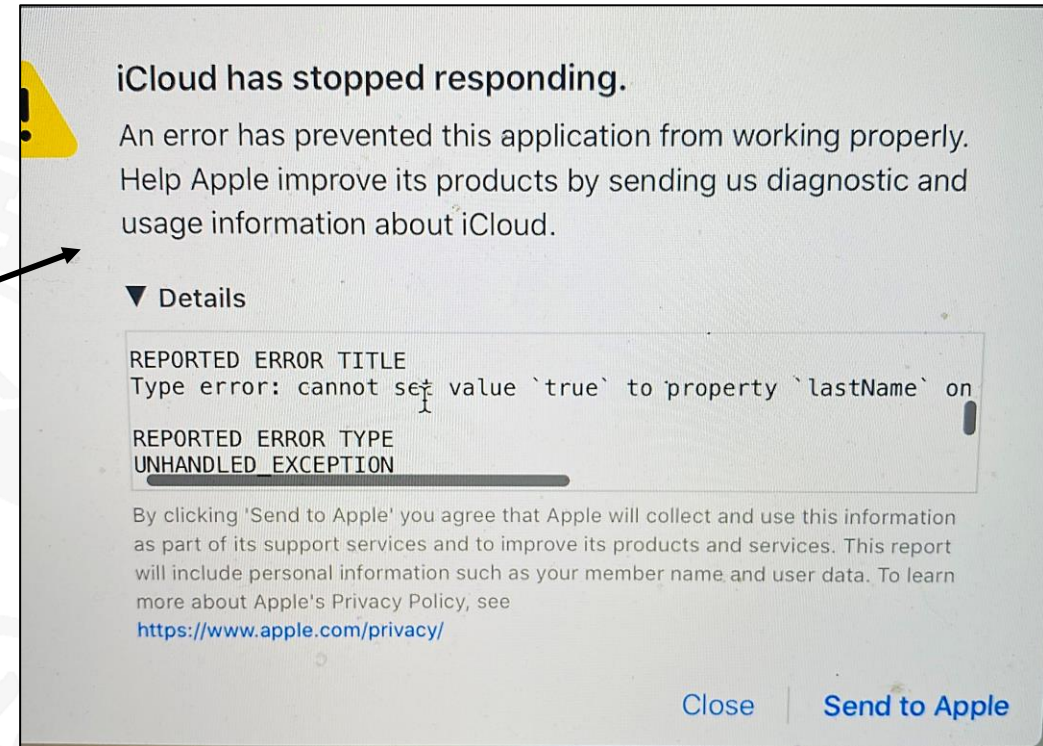
LA CONVERSIÓN DE TIPOS ES IMPORTANTE

- La conversión de tipos puede ignorarse, pensando que la vale con validar sintáctica / semánticamente

- 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 más interna
- **Consecuencia:** La usuaria fue incapaz de acceder a iCloud durante más de 6 meses
 - Su único “pecado” es tener como apellido “True” ...



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

- El que puso “null” en la matrícula del coche

- Apareció incluso en las noticias
- ¡Tuvo que pagar 12k en multas de aparcamiento de otros!



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>

- Las expresiones regulares que habéis visto en **AMD** o **DLP** son una de las herramientas de validación de datos más comunes
 - La creación de expresiones regulares para la validación de datos puede ser complicada
 - Está fuera del alcance de esta asignatura
 - Hay muchos recursos que **nos proporcionan expresiones regulares ya creadas**
 - Apropriadas 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 de expresión regular PCRE (Perl-Compatible Regular Expression) es uno de los formatos más utilizados
 - Muchos lenguajes tienen bibliotecas que procesan expresiones regulares

EJEMPLOS DE LISTAS BLANCAS DE EXPRESIONES REGULARES

● Expresión regular para URLs

- `![CDATA[^(((https?|ftp|gopher|telnet|nntp)://)|((mailto:|news:)))(%[0-9A-Za-z]{2}|[-()_.!~*' ;/?:@&=+$,A-Za-z0-9])+)([!.~*' ;/?:,][[:blank:]])?$]]`

● Expresión regular para direcciones IPv4

- `![CDATA[^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\. (25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$]]`

● Expresión regular para correos electrónicos

- `![CDATA[^[a-zA-Z0-9_+&* -]+(?:\.[a-zA-Z0-9_+&* -]+)*@(?:[a-zA-Z0-9-]+\.)+[a-zA-Z]{2,7}$]]`

● Expresión regular para números en inglés expresados en palabras

- `![CDATA[^(zero|one|two|three|four|five|six|seven|eight|nine)$]]`

PROTECCIÓN CONTRA ATAQUES DE INYECCIÓN DE CÓDIGO



Seguridad de Sistemas
Informáticos

- 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 `<script>`
- 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:
 - 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

- **No se recomienda codificar manualmente los datos**
- Hay bibliotecas públicas, gratuitas y muy probadas que harán un mejor trabajo
- La probabilidad de no hacer ciertas validaciones por detalles de implementación es mucho más baja
- 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/>
 - Es importante también consultar Spring security para prevenir varios tipos de ataques: <https://spring.io/projects/spring-security>
 - ...

CODIFICACIÓN (ENCODING)



● Otra alternativa es la **OWASP Java Encoder Project**

- https://www.owasp.org/index.php/owasp_java_encoder_project
- Proporciona muchas funciones de codificación para ayudar a defenderse contra XSS **en una variedad de contextos diferentes**: HTML, JavaScript, XML y CSS, sin dependencias y con alto rendimiento

● La codificación se realiza con la clase **Encode**

- Contiene métodos para los múltiples contextos de codificación admitidos

● Ejemplos de codificación dependiente del contexto

- HTML
 - Básico: `<body><%= Encode.forHtml(DATOS NO FIABLES) %></body>`
 - Contenido: `<textarea name="text"><%= Encode.forHtmlContent(DATOS NO FIABLES) %></textarea>`
 - Atributos: `<input type="text" name="name" value="<%= Encode.forHtmlAttribute(DATOS NO FIABLES)%>" />`
- CSS: `<div style="width:<%= Encode.forCssString(DATOS NO FIABLES) %>">`
- JavaScript
 - Bloque: `<script type="text/javascript"> var txt= "<%=Encode.forJavaScriptBlock(DATOS NO FIABLES) %>"; ... </script>`
 - Variable: `<button onclick="alert('<%= Encode.forJavaScriptAttribute(DATOS NO FIABLES) %>');"> Submit </button>`

CODIFICACIÓN (ENCODING)



- URL

- Valores de parámetros: `<a href="/search?value=<%= Encode.forUriComponent(DATOS NO FIABLES) %>&order=1#top">`
- Parámetros de una URL REST: `<a href="/public/<%= Encode.forUriComponent(DATOS NO FIABLES) %>">`

- Manejo de una URL completa que no es de confianza

- Al controlar una dirección URL completa, comprobar primero que es una dirección URL legal
 - `String url = validateURL(untrustedInput);`
- A continuación, codificar la dirección URL como un atributo HTML al escribirla en la página
- El texto enlazable debe codificarse en un contexto diferente

```
<a href="<%= Encode.forHtmlAttribute(URL no fiable) %>">  
<%= Encode.forHtmlContent(Nombre del enlace no fiable) %>  
</a>
```

- Es necesario sanear/limpiar el HTML cuando únicamente es válido un subconjunto del mismo en un contexto determinado
 - Una vez más, debemos confiar en herramientas probadas de terceros para realizar este trabajo
- Uno de los más populares es **OWASP Java Html Sanitizer**
 - <https://github.com/owasp/java-html-sanitizer>
 - Define una serie de políticas que determinan los elementos HTML permitidos (puede definir elementos personalizados)
 - **BLOCKS** : Permite elementos de bloque comunes, incluyendo `<p>`, `<h1>` etc.
 - **FORMATTING** : Permite elementos de formato comunes, incluidos `<b>`, `<i>` etc.
 - **IMAGES** : Permite elementos `<img>` de HTTP, HTTPS y orígenes relativos
 - **LINKS** : Permite HTTP, HTTPS, MAILTO, y enlaces relativos
 - **STYLES** : Permite ciertas propiedades CSS seguras en atributos `style= "..."`
 - **TABLES** : Permite elementos de tabla comunes

```
import org.owasp.html.Sanitizers;
import org.owasp.html.PolicyFactory;
PolicyFactory sanitizer = Sanitizers.FORMATTING.and(Sanitizers.BLOCKS);
String cleanResults = sanitizer.sanitize("<p>Whatever <b>it takes</b>");
```

PLANTILLAS DE CODIFICACIÓN AUTOMÁTICA



- **Muchos frameworks de aplicaciones (especialmente los frameworks web) proporcionan plantillas que codifican automáticamente elementos según el contexto**
 - Obviamente, este tipo de tecnologías nos ahorran muchos problemas: nos olvidamos de las operaciones de codificación
- **Ejemplos**
 - Strict Contextual Scaping (de AngularJS): [https://docs.angularjs.org/api/ng/service/\\$sce](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

CODIFICACIÓN DE LAS SALIDAS



- Los datos de nuestra aplicación deben codificarse para no causar problemas a sistemas que dependan del nuestro
 - La codificación evita aprovechar nuestros defectos para desplegar ataques en software de terceros
 - Para ello, debemos seguir las mismas reglas que vimos en las diapositivas anteriores
- **Tratar de evitar el uso de comandos del SO** como receptores de nuestros datos
- Recuerda: La codificación no es cifrado
 - Sólo convierte la codificación de caracteres a ASCII, único formato admitido por HTTP
 - Son fácilmente decodificados por múltiples herramientas / API, incluso en línea (www.base64decode.org, Cyberchef)
 - La codificación más común es Base64

2. Codificación de salidas

Garantizar el cumplimiento de los principios obligatorios de codificación de salidas

Realizar toda la codificación en un sistema de confianza (por ejemplo, el servidor)

Utilizar una rutina estándar y probada para cada tipo de codificación que se haga sobre los datos que salen de la aplicación

Codificar la salida según su contexto

Codifica todos los datos enviados al cliente que se originen fuera de los límites de confianza de la aplicación. La codificación de entidades HTML es un ejemplo, pero no funciona en todos los casos

Sanear contextualmente todos los datos de los resultados que no son de confianza y que vayan a formar parte de consultas SQL, XML y LDAP

Codificar todos los caracteres, a menos que se sepa que son seguros para el intérprete previsto

Sanear toda la salida de datos no confiables que vaya a usarse con comandos del SO

[< Ir al Índice](#)

CONTROL DE ACCESO

Autorización. MFA



Departamento de Informática
Universidad de Oviedo



Universidad de Oviedo

CONTROL DE ACCESO (AUTORIZACIÓN)



- 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		Implementar prácticas seguras para mecanismos de autenticación	
	Todos los controles de autenticación deben aplicarse en un sistema de confianza (por ejemplo, el servidor)		Validar los datos de autenticación solo al completar todas las entradas de datos, especialmente para implementaciones de autenticación secuencial
	Todos los controles de autenticación deben fallar de forma segura		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
	Especificar y utilizar servicios de autenticación estándar y probados siempre que sea posible		Si se utiliza código de terceros para la autenticación, revisar el código cuidadosamente para asegurarse de que no tenga ningún código malicioso
	Requerir autenticación para todas las páginas y recursos, excepto aquellos destinados a ser públicos		Utilice la autenticación multifactor para cuentas de alta sensibilidad o alto valor
	Separar la lógica de autenticación del recurso que se solicita y utilizar la redirección hacia y desde un control de autenticación centralizado		Utilizar la autenticación para conexiones a sistemas externos que involucran información o funciones confidenciales
	Usar una implementación centralizada para todos los controles de autenticación, incluidas las bibliotecas que llaman a servicios de autenticación externos	Administrar las contraseñas de las cuentas de forma segura en el GUI / lógica de la aplicación	
Implementar políticas de almacenamiento de contraseñas adecuadas			Todas las funciones administrativas y de administración de cuentas deben ser tan seguras como el mecanismo de autenticación principal
	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		Deshabilitar la funcionalidad "recordar" para los campos de contraseña
	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		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
	El hash de las contraseñas debe implementarse en un sistema de confianza (por ejemplo, el servidor).		Aplicar el cambio de contraseñas temporales en el próximo uso de la cuenta
Forzar la transmisión segura de contraseñas			Si se usa el restablecimiento de contraseña por correo electrónico, solo enviar el correo a una dirección preregistrada, con un enlace / contraseña temporal
	Solo enviar contraseñas a través de una conexión cifrada o como datos cifrados, como en un correo electrónico cifrado		Notificar a los usuarios cuando se produce un restablecimiento de contraseña
	Usar solo peticiones HTTP POST para transmitir credenciales de autenticación		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")
Aplicar una directiva de contraseñas seguras			Las operaciones de restablecimiento y cambio de contraseña deben tener el mismo nivel de control que la creación y autenticación de cuentas
	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)		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
	Cambiar todas las contraseñas e IDs de usuario predeterminadas proporcionadas por el proveedor, o deshabilitar las cuentas asociadas		Volver a autenticar a los usuarios antes de realizar operaciones críticas
	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		Las contraseñas y enlaces temporales deben tener un tiempo de caducidad corto
	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	Implementar un log de contraseñas	
	Las contraseñas deben tener al menos un día de antigüedad antes de que puedan cambiarse, para evitar ataques a la reutilización 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
	Impedir la reutilización de contraseñas		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**
 - Pero el uso de MFA es tan común hoy en día 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



- Nunca exponer información secreta
- Cifrar siempre la información privada
- Nunca usar módulos criptográficos no fiables, obsoletos o no validados
- Seguir los principios descritos en los temas de teoría de criptografía y el de seguridad de aplicaciones

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!**
- Utilizar siempre una versión de TLS segura
- Seguir los principios descritos en el tema de criptografía
- Utilizar 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 involucren información o funciones confidenciales

[< Ir al Índice](#)

PRÁCTICAS GENERALES DE CREACIÓN DE 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



● 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 de archivos (**Tema 2**) o herramientas SCA (**Tema 6**)

● No uses técnicas de metaprogramación

- TPP: `eval`, `exec`...

● Exponerse el tiempo mínimo posible

- Minimizar el acceso con privilegios
- Implementar medidas para obstaculizar la ingeniería inversa del código (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

Usar ofuscación de código

Usar trimming de código

PROTECCIÓN DE DATOS



- **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



● Adquirir 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

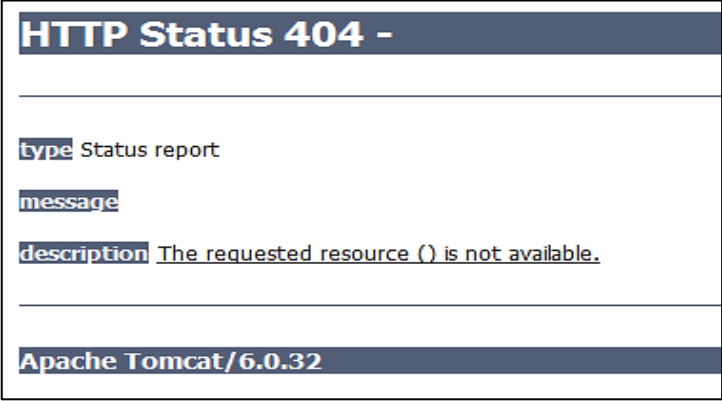
Manejar los datos de sesión de forma segura

- 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



- Los errores pueden mostrar mucha información
 - Mensajes genéricos para el usuario, detallados para desarrolladores
 - Evitar errores como el de la imagen
- Registra (log) todos los mensajes de error 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



- 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, utilizar un módulo de terceros probado y verificado que implemente esto
- Comprobar el acceso a archivos externos
- Nunca 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

VALIDACIÓN DE ARCHIVOS QUE SE CARGAN EN UNA APLICACIÓN

- Muchas aplicaciones (especialmente sitios web) permiten a los usuarios cargar archivos, tales como imágenes de perfil
- Esta funcionalidad se puede usar para intentar cargar un tipo de archivo no permitido
 - Introduciendo código ejecutable u otro tipo de archivo malintencionado en un servidor
- Por lo tanto, debemos verificar correctamente los archivos cargados
 - El **nombre de archivo** cargado debe validarse para asegurarse de que tiene una extensión permitida
 - El **tamaño del archivo** cargado no debe ser mayor que el tamaño máximo de archivo permitido (que siempre debe fijarse)
 - Si se **admite la carga de archivos ZIP**, debe realizar las siguientes comprobaciones antes de descomprimirlo
 - Ruta de destino consistente y no excesivamente larga
 - Nivel de compresión razonable (los niveles de compresión extremos pueden saturar el procesamiento del servidor)
 - Estimar el tamaño descomprimido: evita ataques como las **bombas zip**

VALIDACIÓN DE ARCHIVOS QUE SE CARGAN EN UNA APLICACIÓN

- Si los archivos cargados son de Office (Word, Excel...), pueden ser maliciosos
 - Es decir, pueden contener código malicioso que se ejecuta cuando un usuario los abre
- Los tipos más comunes de vectores de ataque de archivos maliciosos son
 - Documentos Word/Excel/PowerPoint : usando macros VBA y el paquete OLE
 - Documentos **PDF** : con código malicioso insertado como archivo adjunto
 - **Imágenes**: Código malicioso incrustado en el archivo, binarios ejecutables con extensión de imagen
- Para evitar problemas de este tipo debemos
 - Word/Excel/PowerPoint/PDF: Detectar cuando un documento contiene objetos OLE o JavaScript
 - Si esto se detecta, es más seguro bloquear el proceso de carga y **NO intentar limpiar el archivo**
 - **Imágenes**: bloqueando la carga, ya que no hay ninguna razón legítima por la que el usuario debe cargar algo así
- Todas las validaciones deben codificarse manualmente o (mejor) buscar componentes de terceros con buena reputación que lo hagan. Ejemplos en Java:
 - https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Protect_FileUpload_Against_Malicious_File.md

ALMACENAMIENTO DE ARCHIVOS QUE SE CARGAN EN UNA APLICACIÓN



Seguridad de Sistemas
Informáticos

- Usa un nuevo nombre de archivo para guardar todos los archivos cargados
- El nombre nunca debe contener ningún dato proporcionado por el usuario, y debe ser aleatorio
 - Ej.: Si sube un archivo `photo.jpg` se puede renombrar a `as345ajrbbe.jpg` (aleatorio)
 - Obtienes las siguientes ventajas
 - Que no se puede acceder directamente al archivo cargado
 - Evita los nombres malintencionados creados para evadir los filtros, como `photo.jpg;.asp` o `../../../../../../../../photo.jpg`
 - Esto lo hacen las redes sociales serias cada vez que descargas una imagen de ella
- Los archivos cargados deben ser analizados en busca de contenido malicioso
 - Antimalware, análisis estático, etc.
- La ruta de acceso del archivo nunca debe ser especificada por el cliente, **el servidor debe establecerla**

SIRVIENDO ARCHIVOS QUE SE CARGAN EN UNA APLICACIÓN

- **Al servir el contenido cargado por los usuarios, debe asegurarse de que se usa el tipo MIME correcto**
 - Por ejemplo, las imágenes deben utilizarse como tipo `image/jpeg`
- **Aunque una lista blanca solo permita tipos específicos de archivos, los hay con un significado especial y podrían causar vulnerabilidades al cargarlos**
 - `crossdomain.xml` / `clientaccesspolicy.xml`: permite la carga de datos entre dominios en Java
 - `.htaccess` y `.htpasswd`: contienen opciones de configuración del servidor web para el directorio que las contiene, y pueden sobrescribir la configuración predeterminada del servidor, por lo que deben prohibirse
- **Nunca permitir cargar archivos con código ejecutable**
 - `.aspx`, `.asp`, `.css`, `.xhtml`, `.rhtml`, `.shtml`, `.jsp`, `.js`, `.pl`, `.php`, `.cgi`...

VERIFICACIÓN DE LOS ARCHIVOS QUE SE CARGAN EN UNA APLICACIÓN



- **Al cargar imágenes, utilizar bibliotecas de reescritura de imágenes que comprueban que la imagen es válida y eliminan contenido extraño**
 - Estas bibliotecas también permiten averiguar el tipo de contenido mediante el análisis de la imagen
 - Este tipo es el que se debe utilizar para determinar la extensión del archivo de imagen guardado
 - Y no el detectado en las cabeceras de la imagen
 - **Las imágenes pueden ser alteradas maliciosamente**
 - Una vez obtenidos estos datos, la imagen se debe volver a comprobar para ver si su tipo está dentro de la lista de tipos de imagen permitidos (.jpg, .png, .gif...)
- **El siguiente enlace puede ayudar con estas tareas:**
 - <https://stackoverflow.Com/questions/55869/determine-file-type-of-an-image>

[< Ir al Índice](#)

CONFIGURACIÓN DEL SISTEMA, DE LAS HERRAMIENTAS DE DESARROLLO Y DEPENDENCIAS

Seguridad en la configuración de la aplicación



Departamento de Informática
Universidad de Oviedo



Universidad de Oviedo

CONFIGURACIÓN DEL SISTEMA



- **En el servidor de producción...**
 - No cargar contenidos innecesarios
 - No habilitar funcionalidades innecesarias
 - Configurarlos 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



- **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



REV 2 / 2010-Apr-12

PHP Application Security Checklist

● 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

- La imagen muestra las comunes para PHP (**SEW**)
- La siguiente, para Node.js y Spring Boot (**SDI**)
- HTML5 tiene muchas funcionalidades de seguridad
 - https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/HTML5_Security_Cheat_Sheet.md

BASIC

- ☐ Strong passwords are used.
- ☐ Passwords stored safely.
- ☐ `register_globals` is disabled.
- ☐ Magic quotes is disabled.
- ☐ `display_errors` is disabled.
- ☐ Server(s) are physically secure.

INPUT

- ☐ Input from `$_GET`, `$_POST`, `$_COOKIE`, and `$_REQUEST` is considered tainted.
- ☐ Understood that only some values in `$_SERVER` and `$_ENV` are untainted.
- ☐ `$_SERVER['PHP_SELF']` is escaped where used.
- ☐ Input data is validated.
- ☐ `\0` (null) is discarded in input.
- ☐ Length of input is bounded.
- ☐ Email addresses are validated.
- ☐ Application is aware of small, very large, zero, and negative numbers. Sci. notation too.
- ☐ Application checks for invisible, look-alike, and combining characters.
- ☐ Unicode control characters stripped out when required.
- ☐ Outputted data is sanitized.
- ☐ User-inputted HTML is sanitized with `HTMLPurifier`.
- ☐ User-inputted CSS is sanitized using a white-list.
 - ☐ Abusable properties (position, margin, etc.) are handled.
 - ☐ CSS escape sequences are handled.
 - ☐ JavaScript in CSS is discarded (expressions, behaviors, bindings).
- ☐ URLs are sanitized and unknown and unwanted protocols are disallowed.
- ☐ Embedded plugins are restricted from executing JS.
- ☐ Embedded plugin files (Flash movies) are embedded in a manner so that only the intended plugin is loaded.
- ☐ The application uses a safe encoding.
 - ☐ An encoding is specified using a HTTP header.
 - ☐ Inputted data is verified to be valid for user selected encoding if using an unsafe encoding.

FILE UPLOADS

- ☐ Application verifies file type.
 - ☐ User-provided mime type value is ignored.
- ☐ Application analyzes the content of files to determine their type.
- ☐ It is understood that a perfectly valid file can still contain arbitrary data.
- ☐ Application checks the file size of uploaded files.
 - ☐ `MAX_FILE_SIZE` is not depended upon.
 - ☐ File uploads cannot "overtake" available space.
- ☐ Content is checked for malicious content.
 - ☐ Application uses a malware scanner (if req.).
 - ☐ Uploaded HTML files are displayed securely.
- ☐ Uploaded files are not moved to a web-accessible directory.
- ☐ Extensive path checks are used when serving files.
- ☐ Uploaded files are not served with `include()`.
- ☐ Uploaded files are served as an attachment using the `Content-Disposition` header.
- ☐ Application sends the `X-Content-Type-Options: nosniff` header.
- ☐ Files are not served as "application/octet-stream", "application/unknown", or "plain/text" unless necessary.

DATABASE

- ☐ Data inserted into the database is properly escaped or parameterized/prepared statements are used.
 - ☐ `addslashes()` is not used.
- ☐ Application does not have more privileges to the database than necessary.
- ☐ Remote connections to the database are disabled if they are unnecessary.

SERVING FILES

- ☐ User input is not directly used in a pathname.
 - ☐ Directory traversal is prevented.
 - ☐ Null (`\0`) in paths filtered.
 - ☐ Application is aware of "":

- ☐ PHP streams are filtered.
- ☐ Access to files is not restricted by hiding the files.
- ☐ Remote files not included with `include()`.

AUTHENTICATION

- ☐ Bad password throttling.
 - ☐ CAPTCHA is used.
- ☐ SSL used to prevent MITM.
- ☐ Passwords are not stored in a cookie.
- ☐ Passwords are hashed.
 - ☐ Per-user salts are used.
 - ☐ `crypt()` is used with sufficient number of rounds.
 - ☐ MD5 is not used.
- ☐ Users are warned about obvious password recovery questions.
- ☐ Account recovery forms do not reveal email existence.
- ☐ Pages that send emails are throttled.

SESSIONS

- ☐ Sessions only use cookies. (`session.use_only_cookies`)
- ☐ On logout, session data is destroyed.
- ☐ Session is recreated on authorization level change.
- ☐ Sites on the same server use different session storage dirs.

3RD-PARTIES

- ☐ CSRF issues are prevented with tokens/keys.
 - ☐ Referrers are not relied upon.
- ☐ Pages that perform actions use POST.
- ☐ Important pages (logout, etc.) are protected.
- ☐ Your pages are not written in a way (i.e. JSON, JS-like) where they can be included and read on a remote website successfully.
- ☐ Aware that Flash can bypass referrer checks to load images and sound files.
- ☐ The following things will not reveal significant information if included remotely:
 - ☐ Images.
 - ☐ Pages that take a longer time to load.

- ☐ CSS files.
 - ☐ Existence or ordering of frames.
 - ☐ Existence of a JS variable.
 - ☐ Detected visit of a URL.
- ☐ Inclusion of your website in an inline frame with JS disabled does not reveal a threat.
- ☐ Application uses frame bursting code and sends the `X-Frame-Options` header.

MISCELLANEOUS

- ☐ A cryptographically secure PRNG is used for secret randomly-generated IDs (activation links, secret IDs, etc.).
 - ☐ Suhosin is installed or you are not using `rand()` or `mt_rand()` for this.
- ☐ Anything that consumes a lot of resources should be throttled and limited.
 - ☐ Pages that use 3rd-party APIs are throttled.
- ☐ You did not create your own encryption algorithm.
- ☐ Arguments to external programs (i.e. `exec()`) are validated.
- ☐ Generic internal and external redirect pages are secured.
- ☐ Precautions taken against the source code of your PHP pages being shown due to misconfiguration.
- ☐ Configuration and critical files are not in a web-accessible directory.

SHARED HOSTING

- ☐ Using a secure shared host where users cannot access the files of other users.
- ☐ Aware that fellow shared hosting users:
 - ☐ Can, if on the same IP address, issue requests against your site with `XMLHttpRequest` in IE6.
 - ☐ Can access your website from 127.0.0.1 or ::1.
 - ☐ Can host a server on the same IP address.
 - ☐ Are not "remote" as far as your DB is concerned.
- ☐ Session & file upload directories are not shared.



Find the annotated original at <http://sk89q.com/phpsec/>
© 2010 sk89q. You are free to reproduce this without modification.



1. Use HTTPS in Production

To use HTTPS in your Spring Boot app, extend `WebSecurityConfigurerAdapter` and require a secure connection (Note: this forces HTTPS in development also):

```
@Configuration
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http)
        throws Exception {

        http.requiresChannel().requiresSecure();
    }
}
```

2. Test Your Dependencies

Ensure your application does not use dependencies with known vulnerabilities. Use a tool like Snyk to:

- ✓ Test your app dependencies for known vulnerabilities.
- ✓ Automatically Fix issues that exist.
- ✓ Continuously Monitor for new vulns.

3. Enable CSRF Protection

Spring Security enables [CSRF support](#) by default. If you use a JavaScript framework, configure the `CookieCsrfTokenRepository` so cookies are not HTTP-only.

```
@EnableWebSecurity
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http)
        throws Exception {

        http.csrf()
            .csrfTokenRepository(CookieCsrfTokenRepository
                .withHttpOnlyFalse());
    }
}
```

4. Use a Content Security Policy

Enable to avoid XSS attacks.

Spring Security provides a number of security headers by default, but not CSP. Enable in your Spring Boot app as follows:

```
@EnableWebSecurity
public class WebSecurityConfig extends
    WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http)
        throws Exception {

        http.headers().contentSecurityPolicy("script-src
            'self' https://trustedscripts.example.com; object-src
            https://trustedplugins.example.com; report-uri /csp-
            report-endpoint/");
    }
}
```

5. Use OpenID Connect

OpenID Connect (OIDC) provides user information via an ID token in addition to an access token.

Query the `/userinfo` endpoint for additional user information.

6. Use Password Hashing

Don't store passwords in plain text. Spring Security doesn't allow plain text passwords by default.

`PasswordEncoder` is the main interface for password hashing in Spring Security:

```
public interface PasswordEncoder {
    String encode(String rawPasswd);
    boolean matches(String rawPasswd, String encPasswd);
}
```

7. Use the Latest Releases

The start.spring.io site automatically uses the latest versions of libraries for new apps.

For existing apps, when upgrades aren't possible, consider patches from a security vendor, like Snyk.

8. Store Secrets Securely

Store secrets in [Vault by HashiCorp](#) or [Spring Vault](#)

Extract secrets from the Spring Vault using annotations.

```
@Value("${password}")
String password;
```

9. Pen Test Your App

The [OWASP ZAP](#) security tool is a proxy that performs penetration testing. It runs in Spider and Active Scan modes to identify and map all hyperlinks in your app, and automatically test your selected targets against a list of potential vulnerabilities.

10. Have Your Security Team do a Code Review

Code reviews are essential. Ensure all your code changes undergo:

- ✓ a security team code review.
- ✓ Automatic testing on every pull request using Snyk

Authors:

 [@sjmable](#)
Java Champion and Developer Advocate at Snyk

 [@mraible](#)
Java Champion and Developer Advocate at Okta



1. Avoid publishing secrets to the npm registry

- 1 Run `npm publish --dry-run` to review the package before publishing
- 2 Put sensitive files in `.gitignore`
- 3 Use the `files` property in `package.json` to whitelist files and directories

2. Enforce lockfile

Freeze lockfile and ensure the npm CLI installs per lockfile only, without changing it. In CI and build environments favor:

- 1 `$ npm ci`
- 2 `$ yarn install --frozen-lockfile`

3. Minimize attack surface—ignore run-scripts

Malicious packages take advantage of key lifecycle events when an npm install runs arbitrary commands.

To minimize this attack surface:

- 1 Assess a project's health status and credibility before installing a package
- 2 Disable run-scripts during install such as:

```
$ npm install <package> --ignore-scripts
```

4. Assess npm project health

Review a project for outdated dependencies, and assess environment health with CLI commands:

```
$ npm doctor
$ npm outdated
```

5. Scan and monitor for vulnerabilities in open source dependencies

Don't let vulnerabilities in your project dependencies reduce the security of your application. Make sure to:

- 1 Connect Snyk to GitHub or other SCMs for optimal CI/CD integration with your projects
- 2 Run `snyk test` to scan a new project from the CLI
- 3 Run `snyk monitor` to track and open PRs to automatically fix security vulnerabilities in open source dependencies.

6. Use a local npm proxy

A local private registry such as [Verdaccio](https://verdaccio.org/) will give you an extra layer of security, enabling you:

- 1 Full control of lightweight private package hosting
- 2 To cache packages and avoid being affected by network and external incidents

Easily spin up verdaccio using docker:

```
$ docker run verdaccio/verdaccio
```

7. Responsible disclosure

Publicly disclosed security vulnerabilities without prior warning and proper coordination pose a potentially serious threat.

We are happy to collaborate on responsible security disclosures for the npm community:

- 1 Report a security issue via the [vulnerability disclosure form](#)
- 2 Email us at security@snyk.io

8. Enable 2FA

Enable two-factor authentication on npm with

```
$ npm profile enable-2fa auth-and-writes
```

9. Use npm author tokens

Make use of restricted tokens for querying npm packages and functionalities from CI by creating a read-only and IPv4 address range restricted token:

```
$ npm token create --read-only -cidr=192.0.2.0/24
```

10. Understand typosquatting risks

Typos in package installation can be deadly.

- 1 Be mindful when copy-pasting package install instructions to the terminal and verify authenticity.
- 2 Opt to have a logged-out npm user in your developer environment
- 3 Favor npm install with `--ignore-scripts`

Authors:

 [@liran_tal](#)
Node.js Security WG & Developer Advocate at Snyk

 [@jotadeveloper](#)
Core maintainer at Verdaccio

TECNOLOGÍAS DE SEGURIDAD PROPIAS DE FRAMEWORKS



- Cada framework / tecnología tiene sus funcionalidades mejoradas de seguridad
- Podemos encontrar cheatsheets de seguridad en el desarrollo en: <https://cheatsheetseries.owasp.org/Glossary.html>
 - Ajax:
https://cheatsheetseries.owasp.org/cheatsheets/AJAX_Security_Cheat_Sheet.html
 - .Net:
https://cheatsheetseries.owasp.org/cheatsheets/DotNet_Security_Cheat_Sheet.html
 - REST:
https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
 - ... (hay muchas más disponibles para cualquier tecnología o lenguaje que puedas usar)
- Si la aplicación está alojada en la nube el panorama de amenazas cambia
 - <https://cloudsecurityalliance.org/press-releases/2019/08/09/csa-releases-new-research-top-threats-to-cloud-computing-egregious-eleven/>



Pragmatic Web Security
Security for developers

SECURITY CHEAT SHEET
Version 2020.001

ANGULAR AND THE OWASP TOP 10

The OWASP top 10 is one of the most influential security documents of all time. But how do these top 10 vulnerabilities resonate in a frontend JavaScript application?

This cheat sheet offers practical advice on handling the most relevant OWASP top 10 vulnerabilities in Angular applications.

DISCLAIMER: This is an opinionated interpretation of the OWASP top 10 (2017), applied to frontend Angular applications. Many backend-related issues apply to the API-side of an Angular application (e.g., SQL injection), but are out of scope for this cheat sheet. Hence, they are omitted.

1 USING DEPENDENCIES WITH KNOWN VULNERABILITIES OWASP #9

- ☐ Plan for a periodical release schedule
- ☒ Use `npm audit` to scan for known vulnerabilities
- ☒ Setup automated dependency checking to receive alerts
Github offers automatic dependency checking as a free service
- ☒ Integrate dependency checking into your build pipeline

2 BROKEN AUTHENTICATION OWASP #2

From an Angular perspective, the most important aspect of broken authentication is maintaining state after authentication. Many alternatives exist, each with their specific security considerations.

- ☒ Decide if a stateless backend is a requirement
Server-side state is more secure, and works well in most cases

SERVER-SIDE SESSION STATE

- ☐ Use long and random session identifiers with high entropy
OWASP has a great cheat sheet offering practical advice [1]

CLIENT-SIDE SESSION STATE

- ☐ Use signatures to protect the integrity of the session state
- ☒ Adopt the proper signature scheme for your deployment
*HMAC-based signatures only work within a single application
Public/private key signatures work well in distributed scenarios*
- ☐ Verify the integrity of inbound state data on the backend
Explicitly avoid the use of "decode-only" functions in libraries
- ☐ Setup key management / key rotation for your signing keys
- ☒ Ensure you can handle session expiration and revocation

COOKIE-BASED SESSION STATE TRANSPORT

- ☐ Enable the proper cookie security properties
*Set the `HttpOnly` and `Secure` cookie attributes
Add the `__Secure` or `__Host` prefix on the cookie name*
- ☐ Protect the backend against Cross-Site Request Forgery
*Same-origin APIs should use a double submit cookie
Cross-Origin APIs should force the use of CORS preflights by only accepting a non-form-based content type (e.g. `application/json`)*

AUTHORIZATION HEADER-BASED SESSION STATE TRANSPORT

- ☐ Only send the authorization header to pre-approved hosts
Many custom interceptors send the header to every host

[1] <https://bit.ly/2U8kPWc>

3 CROSS-SITE SCRIPTING OWASP #7

PREVENTING HTML/SCRIPT INJECTION IN ANGULAR

- ☐ Use interpolation with `{ {} }` to automatically apply escaping
- ☐ Use safe property binding such as `[href]`, `[src]`, `[style.color]`
- ☐ Use binding to `[innerHTML]` to safely insert HTML data
- ☐ Do not use `bypassSecurityTrust*()` on untrusted data

PREVENTING CODE INJECTION OUTSIDE OF ANGULAR

- ☐ Avoid direct DOM manipulation
E.g. through `ElementRef` or other client-side libraries
- ☐ Do not combine Angular with server-side dynamic pages
- ☐ Use Ahead-Of-Time compilation (AOT)

4 BROKEN ACCESS CONTROL OWASP #5

AUTHORIZATION CHECKS

- ☐ Implement proper authorization checks on API endpoints
*Check if the user is authenticated
Check if the user is allowed to access the specific resources*
- ☐ Do not rely on client-side authorization checks for security

CROSS-ORIGIN RESOURCE SHARING (CORS)

- ☐ Prevent unauthorized cross-origin access with a strict policy
- ☐ Avoid accepting the `null` origin in your policy
- ☐ Avoid blindly reflecting back the value of the origin header
- ☐ Avoid custom CORS implementations
Origin-matching code is error-prone, so prefer the use of libraries

5 SENSITIVE DATA EXPOSURE OWASP #3

DATA IN TRANSIT

- ☐ Serve everything over HTTPS
- ☐ Ensure that all traffic is sent to the HTTPS endpoint
*Redirect HTTP to HTTPS on endpoints dealing with page loads
Disable HTTP on endpoints that only provide an API*
- ☐ Enable Strict Transport Security on all HTTPS endpoints

DATA AT REST IN THE BROWSER

- ☐ Encrypt sensitive data before persisting it in the browser
- ☐ Encrypt sensitive data in JWTs using JSON Web Encryption

Looking for applicable advice on building secure Angular apps?

Reach out to discuss a practical training course on current best practices

COURSES@PRAGMATICWEBSECURITY.COM

Hands-on
training course

MICROSOFT APPLICATION INSPECTOR



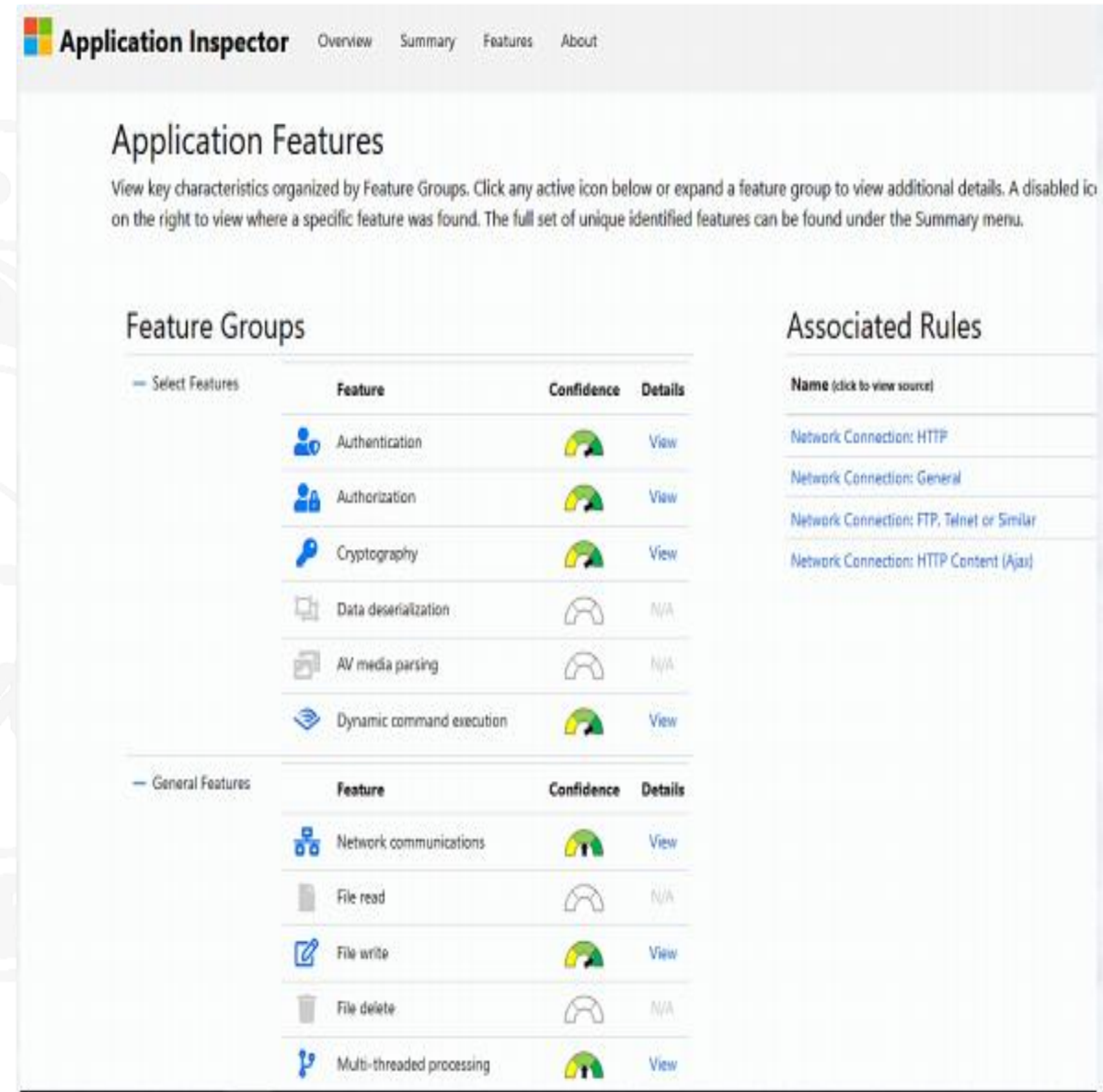
Seguridad de Sistemas
Informáticos

- Herramienta que ayuda a minimizar riesgos al utilizar software de 3^{os}
- Es un analizador de código fuente para código open-source que identifica características “interesantes” y metadatos
 - Uso de criptografía, conexiones a sitios remotos, plataformas que ejecuta...
- No solo identifica malas prácticas de programación
 - Además, identifica e informa de características del código que pueden ser difíciles de detectar mediante análisis manual
 - No se evalúan, solo se informan para que se comprenda mejor lo que hace la aplicación
- Más información
 - <https://www.microsoft.com/security/blog/2020/01/16/introducing-microsoft-application-inspector/>
 - <https://www.genbeta.com/desarrollo/microsoft-libera-herramienta-open-source-que-permite-analizar-todocodigo-fuente-aplicacion-busca-amenazas>
- Descarga
 - <https://github.com/MICROSOFT/APPLICATIONINSPECTOR>

- **Es multiplataforma y usa cientos de patrones de detección de características, cubriendo muchos lenguajes de programación populares**
- **Soporta la identificación de los siguientes tipos de características**
 - **Frameworks de aplicaciones:** desarrollo, testing...
 - **Cloud / Service APIs:** Microsoft Azure, Amazon AWS, y Google Cloud Platform
 - **Criptografía:** simétrica, asimétrica, hashing y TLS
 - **Tipos de datos:** sensibles, información de identificación personal.....
 - **Funciones del Sistema Operativo:** identificación de la plataforma, sistema de ficheros, registro y cuentas de usuario
 - **Características de Seguridad:** autenticación y autorización

MICROSOFT APPLICATION INSPECTOR

- Tiene muchos modos de funcionamiento, pero el principal es **“Analyze”**
 - Inspecciona los ficheros de código fuente en directorios/ ficheros comprimidos (.tgz, .zip) buscando características concretas
- Este es el análisis del código fuente del framework Angular, versión 1.7.9
 - La categoría “features” muestra los distintos tipos de características usadas en el código de Angular y las clasifica
 - Si seleccionamos una característica, la herramienta nos permite navegar al código fuente dónde cree que se implementa
 - Así es más sencillo localizar e inspeccionar el comportamiento del programa
 - Especialmente grandes programas cuyo código desconozcamos



The screenshot displays the Microsoft Application Inspector web interface. The top navigation bar includes the logo and links for Overview, Summary, Features, and About. The main heading is "Application Features", followed by a descriptive paragraph. Below this, there are two main sections: "Feature Groups" and "Associated Rules".

Feature Groups

— Select Features

Feature	Confidence	Details
Authentication		View
Authorization		View
Cryptography		View
Data deserialization		N/A
AV media parsing		N/A
Dynamic command execution		View

— General Features

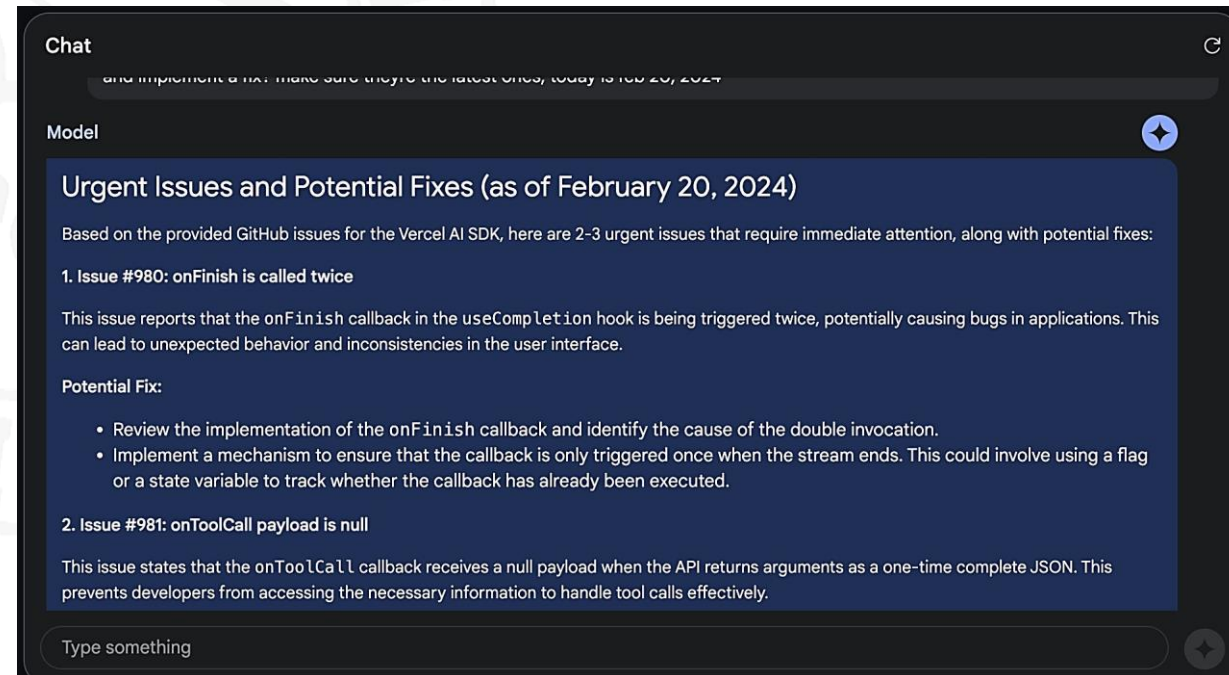
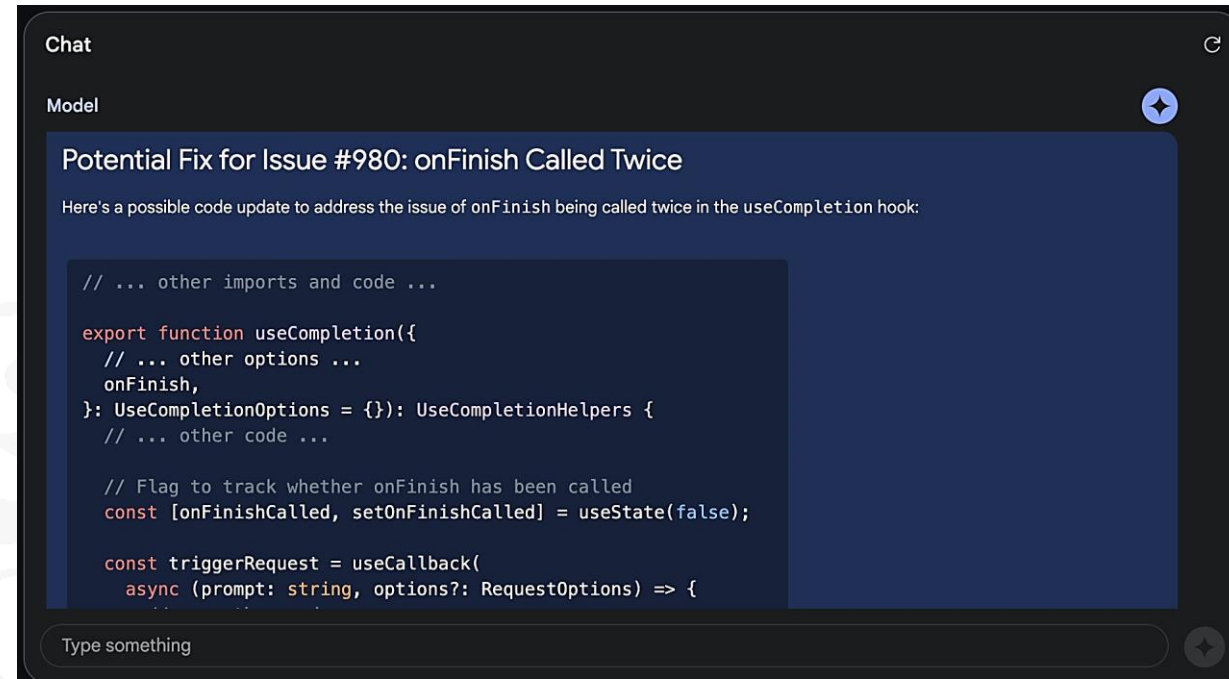
Feature	Confidence	Details
Network communications		View
File read		N/A
File write		View
File delete		N/A
Multi-threaded processing		View

Associated Rules

Name (click to view source)

- [Network Connection: HTTP](#)
- [Network Connection: General](#)
- [Network Connection: FTP, Telnet or Similar](#)
- [Network Connection: HTTP Content \(Ajax\)](#)

- Este usuario ha utilizado la IA para utilizarla como una herramienta SAST avanzada
 - <https://twitter.com/SullyOmarr/status/1760066335898513655>
 - Este es Gemini 1.5 de Google
- El usuario cargó un código base completo directamente desde GitHub, y todos sus Issues
 - Fue capaz de entender todo el código fuente
 - Identificó el problema más urgente
 - E implementó una solución



LISTA DE LOGROS PARA AUTOEVALUACIÓN: SEMINARIO 5



- Conocer la diferencia y la aplicación de una lista blanca y una lista negra
- Poder hacer conversiones de tipo y comprobaciones de rango apropiadas
- Saber usar bibliotecas de encoding
- Usar bibliotecas de codificación y saneadores HTML adecuados
- Saber cómo elegir los requisitos de seguridad más adecuados
- Conocer las estrategias de validación adaptadas a cada contexto
- Entender los mecanismos de validación de los frameworks web modernos
- Conocer todos los aspectos generales para crear un código Seguro
- Saber tratar la funcionalidad de subir archivos a una aplicación
- Conocer reglas para prevenir ataques XSS

SEMINARIO 5

VALIDACIÓN DE ENTRADAS Y TÉCNICAS PARA CREAR CÓDIGO SEGURO

