UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja (UNIR)

Escuela de Ingeniería

Máster universitario en Dirección e Ingeniería de Sitios Web

RAMA PROFESIONAL

Auditoría de Seguridad sobre un WCMS privativo propiedad de una PYME española

Trabajo Fin de Máster

Presentado por: Castillo Robles, Francisco

Director/a: Cobo Martín, Manuel Jesús

Ciudad: León (España)
Fecha: 23/09/2015

Agradecimientos

Con este trabajo finaliza un intenso curso de estudio, durante el cual he sido capaz de compatibilizar las tareas del máster, un bonito periodo de prácticas en empresa, mi familia, mis amigos y mis aficiones.

Agradecer principalmente a mis padres que, con el esfuerzo y tesón de su día a día, me han ido inculcando a lo largo de mi vida una educación y unos valores intachables. La ayuda constante que he recibido siempre por parte de ellos ha hecho que todos mis objetivos y decisiones hayan llegado a buen puerto hasta día de hoy. Pero sobretodo, agradecerles en particular el esfuerzo realizado este último año, para que pudiese estudiar este máster en los tiempos difíciles que corren.

Agradecer a mi hermana todo el apoyo y comprensión que me brinda día a día. Tanto en los buenos, como sobretodo en los malos momentos, ella siempre está ahí conmigo.

Por último, me gustaría agradecerle también a mi tutora del máster, María Poza, su apoyo y cercanía durante todo este periodo, dando la cara siempre por sus alumnos tanto en las buenas como en las malas. Así mismo, mi gratitud hacia Manuel Jesús Cobo, director de este trabajo fin de máster, por su comprensión, ánimos y ayuda constante. Y al resto de docentes, que tanto han ido aportándome día a día en esta inolvidable experiencia educativa, mi más sincero agradecimiento también.

Contenido

Índice de figuras

Índice de tablas

Relación de acrónimos

- ✓ **WCMS:** Web Content Management System
- ✓ **S.L.:** Sociedad Limitada
- ✓ **PYME:** Pequeña Y Mediana Empresa
- ✓ **SME:** Small and Medium Enterprise
- ✓ **OWASP:** Open Web Application Security Project
- ✓ **SI:** Sistema Informático
- ✓ **E/S:** Entrada/Salida
- ✓ **OS:** Operating System
- ✓ **SEO:** Search Engine Optimization
- ✓ **ISO:** International Organization for Standardization
- ✓ **URL:** Uniform Resource Locator
- ✓ **PHP:** Hypertext Pre-Processor
- ✓ **AJAX:** Asynchronous JavaScript And XML
- ✓ **SQL:** Structured Query Language
- ✓ **CSS:** Cascading Style Sheets
- ✓ **HTML:** HyperText Markup Language
- ✓ **XML:** eXtensible Markup Language
- ✓ **IIS:** Internet Information Services
- ✓ **HTTP:** HyperText Transfer Protocol
- ✓ **SaaS:** Software as a Service
- ✓ **LDAP:** Lightweight Directory Access Protocol
- ✓ **XSS:** Cross-Site Scripting
- ✓ **CSRF:** Cross Site Request Forgery
- ✓ **SDLC:** Systems Development Life Cycle
- ✓ **DoS:** Denial Of Service
- ✓ **API:** Application Programming Interface
- ✓ **SSL:** Secure Sockets Layer
- ✓ **TLS:** Transport Layer Security
- ✓ **CSP:** Content Security Policy
- ✓ **DBMS:** Data Base Management System
- ✓ **UI:** User Interface
- ✓ **CAPTCHA:** Completely Automated Public Turing test to tell Computers Humans Apart
- ✓ **IP:** Internet Protocol
- ✓ **ZAP:** Zed Attack Proxy

Resumen

El presente trabajo se basa en la realización de una auditoría de seguridad informática sobre un sistema gestor de contenidos web (WCMS) privativo, no comercial, perteneciente al departamento de desarrollo web de una PYME española: *Soluciones Informáticas Santa Mónica, S.L.*

El propósito principal de este trabajo es analizar de forma focalizada el software en cuestión en busca de posibles vulnerabilidades, a través de una selección de técnicas y mecanismos pertinentes, orquestado todo ello por la "*Guía de pruebas OWASP 2008 v3.0*".

Como resultado, se espera conseguir que la empresa propietaria de dicho software adopte las medidas necesarias, recomendadas de forma justificada en los últimos capítulos, a fin de mejorar e incrementar los niveles de seguridad y reducir, consecuentemente, posibles ataques malintencionados de terceros sobre este sistema gestor de contenidos web.

Palabras Clave: Auditoría de Seguridad – WCMS – OWASP

Abstract

The aim of this project is to conduct computing security audit on a web content management system (WCMS) proprietary, non-commercial, from the department of development of Spanish SMEs: *Santa Monica Computer Solutions, SL*.

The main purpose of this paper is to analyze in targeted software in question for possible vulnerabilities, through a selection of techniques and mechanisms, all orchestrated by the "*2008 OWASP Testing Guide v3.0*".

As a result, it is expected to get the company that owns the software take the necessary measures, justifiably recommended in later chapters, in order to improve and increase safety levels and reduce, consequently, possible malicious attacks by third parties on this web content management system.

Keywords: Security Audit – WCMS – OWASP

Capítulo 1: Introducción

La **Seguridad Informática** [1] garantiza que los recursos informáticos de una entidad se encuentren disponibles para cumplir sus objetivos, es decir, que no se vean alterados o dañados por factores externos.

En términos generales, la seguridad podemos entenderla como "*aquellas reglas técnicas y/o actividades destinadas a prevenir, proteger y resguardar lo que se considera como susceptible de robo, daño o pérdida, ya sea de forma personal, grupal o empresarial*". No se trata de un producto, sino de un proceso. En este sentido, es **la información** el elemento primordial que debemos proteger. Pero, ¿de quién debemos protegernos? Pues de personas ajenas a esta información, comúnmente conocidas como piratas informáticos o hackers, que buscan tener acceso a una determinada red empresarial o SI para modificar, sustraer o borrar datos.

Para esta auditoría en concreto nos interesa centrarnos un poco más en una rama específica de la seguridad informática, como lo es la **seguridad de entornos web** [2], la cual se encarga particularmente de la seguridad de sitios, aplicaciones y servicios web. El crecimiento exponencial que ha adquirido Internet en las últimas décadas trae consigo un efecto secundario muy sensible: la **privacidad de la información** [3], tanto a nivel personal como profesional. Hoy en día, en Internet encontramos un sinfín de comercios online, negocios que hacen circular cantidades de dinero exorbitantes, redes de servicios que habilitan el comercio a nivel internacional, así como un elevado número de redes sociales que albergan información sensible de la vida privada de sus miembros.

Según el punto de vista, podemos creer sin dudar que los puntos más delicados de la seguridad en Internet recaigan sobre los servidores web, considerados puntos de intervención directa con las masas de usuarios, y los lenguajes de programación con los que codificamos las aplicaciones que se ejecutarán posteriormente en estos servidores web. Sin embargo, es un hecho demostrado a lo largo de los años, que la mayoría de los problemas detectados en servicios web no son provocados por fallos intrínsecos de ninguna de estas partes, ya que una inmensa cantidad de estos se producen por **malos usos y/o costumbres por parte de los programadores**, principalmente por la escritura defectuosa o deficiente de código.

Debemos asimilar que programar aplicaciones web seguras no es una tarea sencilla, que requiere por parte del programador no solo prestar atención en cumplir con el propósito

funcional básico de estas, sino una concepción global de los peligros o riesgos que puede correr la información contenida, recibida y solicitada por el sistema en cuestión.

En la actualidad, aunque existen infinidad de publicaciones que permiten formar un criterio sobre este tema, no existen acuerdos básicos oficiales sobre lo que se debe o no se debe hacer, y lo que en algunas publicaciones se recomienda, en otras es refutado. Sin embargo, en lo sustancial, sí que existen algunas sugerencias o recomendaciones que son "universales" y que serán estas las que describamos tomando, como referencia las pautas y políticas de seguridad marcadas por la **Guía de pruebas OWASP v3.0**, estándar de facto para la seguridad de entornos web.

Por último, antes de pasar a detallar los objetivos generales que rigen esta auditoría, me gustaría hacer una breve reseña sobre otro concepto muy vinculado a la seguridad informática. Se trata, en este caso, del concepto de **vulnerabilidad**. En líneas generales, entendemos por vulnerabilidad [4] "*la exposición implícita a un riesgo"*, que trasladado a un ámbito informático, ésta es "*un punto débil del software que posibilita que un atacante comprometa la confidencialidad, disponibilidad o integridad del mismo*".

1.1. Objetivos generales

Como aportación general, con este trabajo se pretende aplicar una auditoría de seguridad informática sobre un "Sistema Gestor de Contenidos Web" – en adelante WCMS –. Este software, de carácter privativo en propiedad de una PYME española, posee numerosos elementos, características y/o funcionalidades potencialmente vulnerables, y nuestra principal misión sobre él será:

- Analizar su código fuente comprobando la calidad de implementación.
- Describir justificadamente las fallas de seguridad detectadas sobre éste.
- Plantear las soluciones pertinentes, recomendadas por la OWASP, a fin de mejorar e incrementar los niveles de seguridad del sistema a evaluar.

1.2. Estructura del trabajo

La presente memoria de Trabajo Fin de Máster se estructura en los siguientes capítulos:

- **Capítulo 2**

Denominado “Marco teórico”, se describirá, por un lado, el contexto y el estado del arte del sistema a evaluar; presentando las características del propio sistema, los elementos concretos a medir, los motivos por los cuales nos centramos en este sistema en particular, etc. Y, por otro lado, se desglosarán las hipótesis u objetivos específicos del trabajo, a fin de conocer su efecto observable.

- **Capítulo 3**

Denominado “Metodología de evaluación de seguridad”, se explicará la metodología utilizada para alcanzar los objetivos marcados; qué pasos se van a seguir dentro de la planificación de la auditoría y por qué, qué instrumentos, fuentes y tecnologías se van a emplear para ello, cómo se van a analizar las posibles vulnerabilidades detectadas, etc. Finalmente, este capítulo albergará un apartado sustancial en el cual se describirán y se llevarán a cabo una serie de test y pruebas pertinentes sobre el software a auditar.

- **Capítulo 4**

Denominado “Resultados”, se realizará una descripción de la contribución del trabajo donde figuren tablas de resumen, identificación de datos relevantes, gráficas de resultados, etc. De cualquier forma, se tratará de una exposición objetiva sin valoraciones ni justificaciones. Para finalizar el capítulo, se abrirá un apartado de resolución donde figurarán los problemas hallados y se propondrán las intervenciones de solución y mejora pertinentes.

- **Capítulo 5**

Denominado “Conclusiones”, se redactará un resumen final donde quede reflejado el problema que se ha tratado, cómo se ha abordado, la justificación de la solución y la relevancia y alcance de la aportación de este trabajo. También se llevará a cabo una relación de los resultados obtenidos con los objetivos planteados y se incluirán unas líneas predecibles de investigación y desarrollo, donde figuren las perspectivas de futuro y la posibilidad de aportar valor añadido al trabajo.

- **Bibliografía y Anexos**

En estas dos secciones quedarán recopilados todas las referencias bibliográficas empleadas y los ficheros adjuntos necesarios para la correcta realización del presente trabajo.

Capítulo 2: Marco teórico

En este capítulo comenzaremos exponiendo la situación actual que atraviesan los sistemas gestores de contenido web a nivel global y la importancia que exige mantener seguros este tipo de sistemas.

En primer lugar, nos centraremos en algunos de los aspectos clave que se tratarán en este Trabajo Fin de Máster. En particular, se responderán a las siguientes cuestiones: ¿Qué es un sistema gestor de contenido web o WCMS? ¿Para qué se utiliza? ¿Por qué elegimos uno y no otro?

Un WCMS [5] es "*un software que proporciona autorías de sitios web, colaboración y herramientas de administración que permiten a los usuarios crear y gestionar contenidos digitales para la web*". Hay que destacar que la mayoría de estos sistemas se sirven de una base de datos o un repositorio de contenidos donde se almacena toda la información necesaria para dichos sistemas; metadatos, contenidos de E/S del sitio web, activos de información, etc.

Este tipo de sistemas está diseñado frecuentemente para permitir que usuarios no técnicos puedan crear y gestionar sus propios sitios web con relativa facilidad, apoyándose en interfaces basadas en navegador con las cuales la administración resulta mucho más liviana e intuitiva. Estos sistemas permiten así definir a sus editores los contenidos del sitio, su estilo visual, configurar ciertos aspectos de la funcionalidad, etc. Sin embargo, no todos los WCMS admiten a este perfil de usuario no técnico y requieren de uno con mayor nivel de conocimiento sobre desarrollo web, para configurar e implementar características y funcionalidades al sistema, como es el caso del WCMS que auditaremos en los próximos apartados.

Cabe destacar una serie de **ventajas genéricas** a la hora de usar un WCMS:

- Mayor vivacidad en el desarrollo web.
- Amplia capacidad de personalización.
- Facilita el soporte, la gestión y el mantenimiento de sitios web dinámicos.
- Transmite coherencia entre el diseño, la navegación y los contenidos.

Dependiendo del WCMS que estemos utilizando, la clase de contenidos con la que trabajaremos puede ser muy variopinta; páginas web, artículos, entradas de blog, tareas, productos, documentos, cursos, etc. Es por ello que, el tipo de contenido es un aspecto

fundamental para seleccionar el modelo de WCMS que mejor se adapte a nuestras necesidades.

Algunos **criterios de selección importantes** para un WCMS suelen ser:

- Objetivo del sitio web: ¿a quién va destinado?, requisitos técnicos y funcionales, etc.
- Propiedades generales: capacidad tecnológica y grado de personalización/extensión.
- Tipo de licencia: Freeware, Open Source, software comercial, trial, etc.
- Arquitectura: tipo de relación entre "contenido-presentación-estructura".
- Facilidad de manejo (usabilidad).
- Actualización de versiones.

Llegado este punto, donde debemos tener claros nuestros criterios particulares, tomaremos la decisión final para la elección de un WCMS que se ajuste finalmente a nuestras necesidades específicas, disponiendo de multitud de **categorías** entre las que poder elegir; foros, blogs, wikis, e-commerce, educación, portales de contenido múltiple, etc.

A continuación, se presenta una relación de los WCMS más populares dentro de las categorías más relevantes:

- Foros → phpBB – MyBB
- Blogs → WordPress – Open Blog
- Wikis → MediaWiki – DokuWiki
- E-Commerce → Magento – PrestaShop
- E-Learning → Moodle – eFront
- Gestión de proyectos → qdPM – Feng Office
- Portales → Joomla! – Drupal
- Social / Comunidad → Dolphin - elgg

En cuanto a la situación actual que atraviesa el uso o empleo de este tipo de sistemas, podemos decir que [6] en los últimos años han proliferado de forma muy significativa, algo razonable ya que han supuesto una buena forma de que usuarios sin conocimientos técnicos de programación, diseño y maquetación, puedan crear su propia web y actualizarla de forma rápida y sencilla. A día de hoy los WCMS más utilizados son *WordPress*, *Joomla!* y *Drupal*, siendo *WordPress* el que mayor evolución ha presentado al pasar de ser un gestor de uso exclusivo para blogs, a emplearse para desarrollar todo tipo de sitios web.

Sin embargo, siempre tendremos la otra cara de la moneda, donde se abre el eterno debate de: *¿es mejor usar un WCMS o crear una web desde cero?* Ya hemos hablado un poco de las ventajas que presenta el uso de los WCMS pero, por ejemplo [6], para un diseñador o un programador web, desarrollar una web desde cero, aunque muchos lo vean como una tarea excesivamente complicada, les va a permitir un grado de personalización infinitamente mayor que utilizando un WCMS.

Sin duda el equilibrio de esta balanza está servido, hay que tener en cuenta que también existen otros muchos factores que influyen a la hora de tomar esta decisión; posicionamiento SEO, la seguridad, el diseño, la gestión de usuarios, etc. Por lo tanto, no es una decisión que debamos tomarnos a la ligera, ni escatimar mucho en gastos si carecemos de los conocimientos técnicos necesarios como para depender de nosotros mismos.

Por último, antes de pasar a presentar detalladamente el WCMS que auditaremos, es de trascendental importancia mencionar y tener en cuenta [7] las **implicaciones de seguridad** en el uso de los gestores de contenido web. Multitud de usuarios y entidades hacen uso de los WCMS pero no tienen en cuenta, por lo general, las medidas básicas de seguridad, uno de los aspectos más controvertidos derivados a la hora de utilizar este tipo de herramientas. Los fallos en los WCMS generalmente son problemas originados por; errores en la administración, tanto del propio WCMS como del sistema que lo aloja (hosting), la instalación de módulos/plugins de terceros y/o errores de programación del propio gestor del sistema.

A continuación, se presentan una serie de **acciones o medidas aconsejables** [7] a tener en cuenta en relación con la seguridad de estos entornos:

- Elegir un servicio de hosting fiable.
- Configurar los servidores de forma segura: establecer parámetros de control con los permisos y accesos al servidor, así como con las páginas que éste sirve.
- Utilizar, en la medida de lo posible, versiones actualizadas tanto de nuestro WCMS como de nuestro OS en general.
- Abstenerse de instalar en nuestro WCMS módulos/plugins de terceros que no hayan sido auditados previamente.
- Implementar férreas políticas de seguridad a la hora de administrar nuestro portal.
- Suscribirse a foros de seguridad, principalmente de nuestro WCMS, a fin de mantenernos al día sobre posibles vulnerabilidades descubiertas en estos sistemas.

2.1. Contexto y estado del arte del sistema a evaluar

El siguiente apartado de este capítulo nos va a exponer ya de una forma más profunda y detallada el WCMS que vamos a auditar.

A continuación se detallará brevemente la empresa propietaria del software que evaluaremos. ***Soluciones Informáticas Santa Mónica, S.L.*** es una PYME española enmarcada en la Comunidad Autónoma de Castilla y León, concretamente en la provincia de León. Nace como empresa en diciembre de 1997 y posee a sus espaldas una amplia experiencia en desarrollo de aplicaciones para la PYME, mantenimiento de sistemas y desarrollo web. Sus actividades principales se desempeñan sobre los departamentos de programación, sistemas y redes e imagen y desarrollo web.

Centrándonos en el departamento de imagen y desarrollo web de esta empresa, destacaremos su metodología de trabajo. En líneas generales, este departamento capta clientes de índole y envergadura muy diversa; desde clientes particulares, hasta grandes instituciones, pasando por pequeñas y medianas empresas. Dentro de esta metodología de trabajo que mencionamos, cabe destacar como elemento o herramienta primordial el WCMS que utilizan para desarrollar todos sus sitios web con un **nivel de personalización totalmente a la carta**.

Este sistema gestor de contenidos web es un producto desarrollado completamente por el departamento de imagen y desarrollo de la citada empresa. Es considerado, por tanto, un software totalmente privativo, el cual no se ha enfocado nunca para su comercialización sino para su propio autoconsumo, que les permita poder llevar a cabo todos sus desarrollos web de una forma lo más organizada, personalizable y cómoda posible.

La utilización de este software requiere previamente de conocimientos técnicos avanzados en programación web. Para que nos hagamos una idea, cuando un cliente contrata un desarrollo web por parte de esta empresa, ésta lo que hace es ir programando desde cero todos los requisitos exigidos e implementando un **panel de control a medida**, para que posteriormente el propio cliente pueda gestionar todos los contenidos de su sitio web que hayan sido previamente acordados en la fase de negociación. Es decir, todos los clientes van a poseer un panel de control similar en cuanto a interfaz gráfica y estilo, pero exclusivo en cuanto a los contenidos personalizables que requiera cada cliente en particular y sus necesidades de uso y/o negocio.

Veamos, poniendo como ejemplo **(de aquí en adelante)** la web de la propia empresa "*Soluciones Informáticas Santa Mónica, S.L.*": **http://www.smonica.com**

Accediendo al enlace anterior podemos observar lo que a simple vista es la **página principal** de este sitio web:

Figura 2.1.1. – Página principal de la web http://www.smonica.com

Para **acceder al panel de control** que gestiona el contenido del sitio web, tanto para este como para el resto de los sitios web que se desarrollan en esta empresa, debemos añadir "**/admin**" a la URL de nuestra web: **http://www.smonica.com/admin**

Figura 2.1.2. – Acceso al 'Panel de Control' de la web de Santa Mónica

Una vez introduzcamos '*Usuario*' y '*Contraseña*' accederemos al gestor de contenidos desde donde podremos personalizar nuestro sitio web a través de las diferentes secciones que lo componen; Textos, Img. Empresa, Carrusel, etc.

Figura 2.1.3. – Página de Inicio del WCMS de la web de Santa Mónica

Vamos a ver un sencillo ejemplo de uso y funcionamiento. Pongámonos en el caso de que queramos modificar la "*Política de Privacidad*" de nuestro sitio web. Pues de forma fácil e intuitiva accederemos a la sección de '*Textos*' y, seleccionando la opción de '*Política de Privacidad*', podremos editar el contenido de este apartado, tal y como se puede observar en la siguiente figura, y **verificar los cambios realizados en nuestra web en tiempo real**:

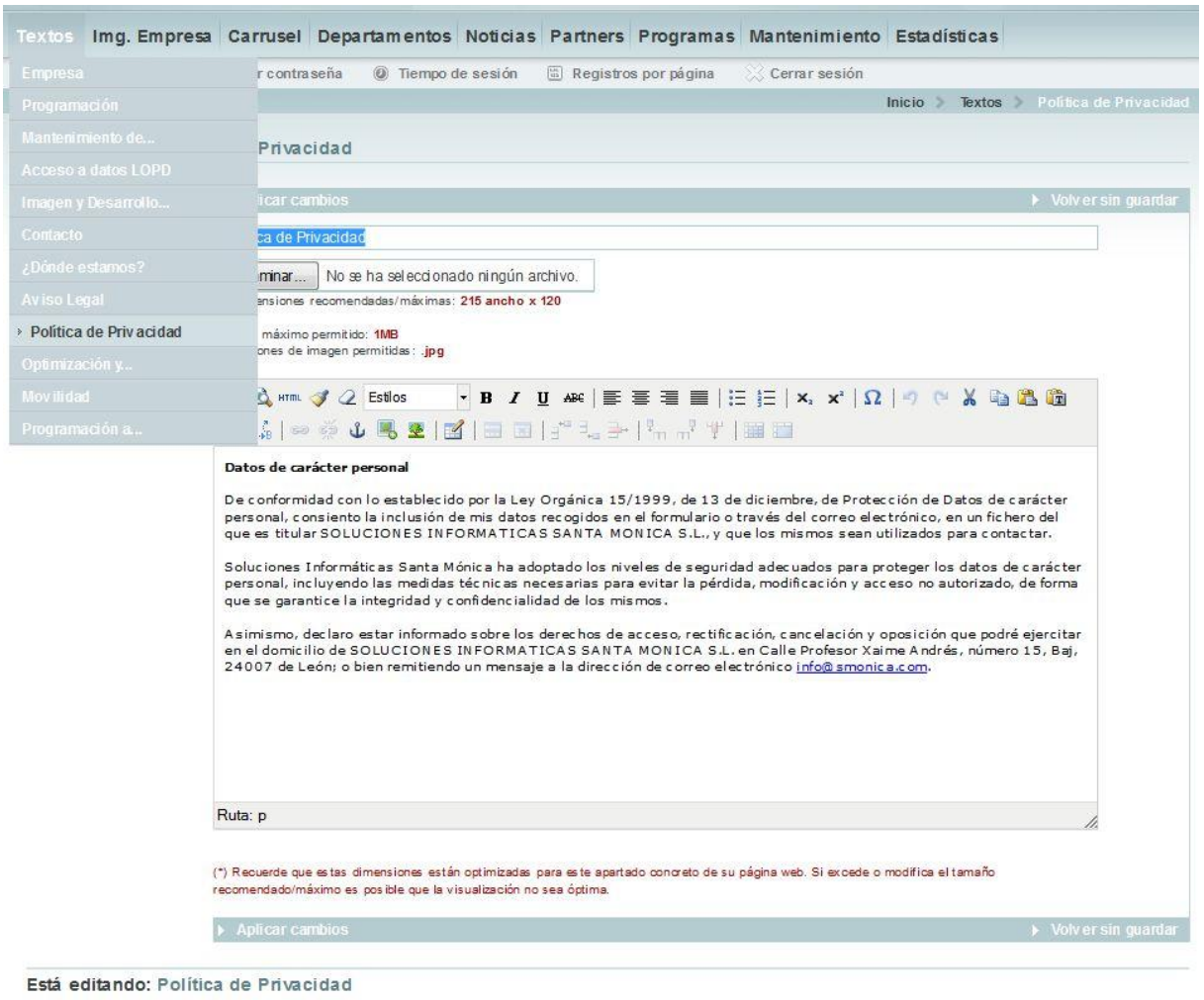

Figura 2.1.4. – Ejemplo de funcionamiento: edición de una sección de texto

Hasta aquí todo aparenta fácil y sencillo, y en realidad para un cliente que ha configurado a conciencia todos los aspectos que desea personalizar dentro del sitio web contratado, lo es. Sin embargo, la otra cara de la moneda es una tarea bastante más compleja para un usuario no técnico. A fin de cuentas, aunque la mayoría de los sitios web compartan un alto porcentaje de componentes, se está implementando un WCMS específico para cada cliente. Por este motivo, **no podremos auditar un WCMS genérico** como tal, al estilo de *Joomla!* o

WordPress, sino que **debemos centrarnos en un caso particular**, que en esta ocasión será el propio WCMS de la web de "*Soluciones Informáticas Santa Mónica*", la misma que hemos tomado como referencia en los ejemplos anteriores. No obstante, como bien ya he mencionado, la mayoría de estos WCMS específicos comparten entre sí una serie de módulos, librerías y características comunes, que será en lo que nos centremos principalmente a la hora de llevar a cabo esta auditoría.

Para hacernos una idea, a vista de directorios y ficheros, un sitio web implementado en el departamento web de esta empresa presentará el siguiente aspecto:

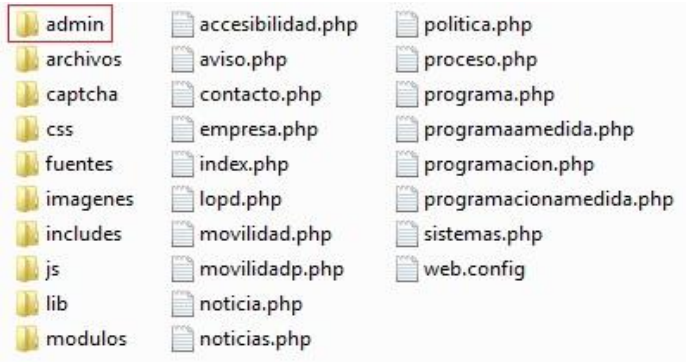

Figura 2.1.5. – Ejemplo de un sitio web a vista de directorios y ficheros

Debemos prestar especial atención a la primera carpeta: **admin**. Este directorio contiene todos los directorios y ficheros necesarios que componen el WCMS (Panel de Control) correspondiente al sitio web que se esté implementando. Echémosle un vistazo:

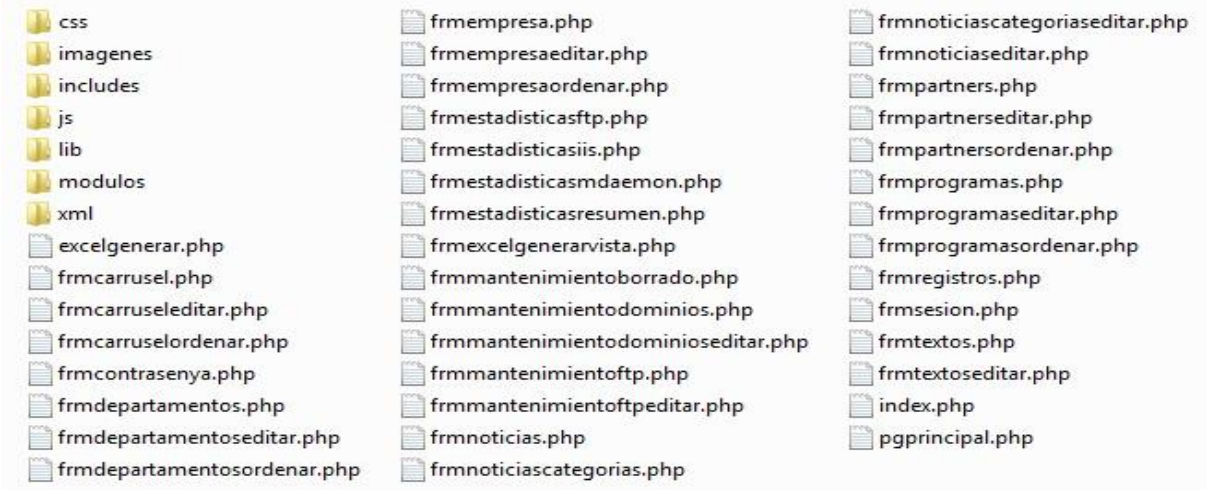

Figura 2.1.6. – WCMS correspondiente a la web de Santa Mónica

Como podemos observar, estamos ante un WCMS muy particular y exclusivo, siendo estos los principales motivos para la elección de auditar este software y no otro.

Para concluir este apartado, vamos a exponer las **características técnicas más destacadas** a la hora de desarrollar cada WCMS y los sitios web vinculantes:

- Está escrito en código abierto, concretamente se utiliza el lenguaje de programación **PHP**, en el lado del servidor.
- En el lado del cliente, se utiliza el lenguaje de programación **JavaScript**, con destacada presencia de la biblioteca **jQuery** y la técnica de desarrollo web **AJAX**.
- Cada sitio web se sirve de una **base de datos SQL** que interacciona dinámicamente con el WCMS correspondiente a dicha web.
- En cuanto a la presentación se utilizan hojas de estilo **CSS** y para su estructura los lenguajes de marcado **HTML** y **XML**.
- En cuanto al alojamiento, "*Soluciones Informáticas Santa Mónica, S.L.*" trabaja tanto con servidores web **IIS** como con **HTTP Apache**.

2.2. Objetivos específicos de investigación/desarrollo

En el siguiente apartado vamos a tratar de focalizar las hipótesis u objetivos específicos, a fin de conocer el efecto observable que pueden producir las fallas de seguridad en el software que se pretende auditar.

Debemos tener en cuenta que existen numerosas posibles vulnerabilidades en entornos y aplicaciones web y que de éstas pueden derivar otras a mayores, y así sucesivamente. Es por ello, que vamos a centrarnos en lo que se conoce como: **OWASP Top 10**.

OWASP Top 10 es un documento que representa un consenso a nivel mundial sobre las diez vulnerabilidades o riesgos de seguridad más críticos en entornos y aplicaciones web. **No es un programa de seguridad web**. Este documento se actualiza y se publica cada 3 años por la organización OWASP (última actualización en 2013), y "*está basado en 8 conjuntos de datos de 7 firmas especializadas en seguridad de aplicaciones, incluyendo 4 empresas consultoras y 3 proveedores de herramientas SaaS. Estos datos cubren más de 500.000 vulnerabilidades a través de cientos de entidades y miles de aplicaciones. Las vulnerabilidades del Top 10 son seleccionadas y priorizadas de acuerdo a estos datos de prevalencia, en combinación con estimaciones consensuadas de explotabilidad, detectabilidad e impacto*".

El objetivo fundamental de este documento es, sin ir más lejos, el de **crear conciencia y educar** desarrolladores, diseñadores, arquitectos, gerentes y organizaciones, sobre las consecuencias de las vulnerabilidades web más destacadas. No se trata de ningún tipo de normativa ISO o estándar que haya que cumplir de forma rigurosa a la hora de desarrollar entornos y aplicaciones web.

A continuación se resumen las vulnerabilidades del actual documento OWASP Top 10, como objetivo específico de investigación y desarrollo a la hora de auditar el WCMS previamente descrito:

Amenaza	Descripción
A1 –Inyección	Los fallos de inyección, tales como SQL, OS, y LDAP, ocurren cuando datos no confiables son enviados a un interprete como parte de un comando o consulta. Los datos hostiles del atacante pueden engañar al interprete en ejecutar comandos no intencionados o acceder datos no autorizados.
A2 – Pérdida de Autenticación y Gestión de Sesiones	Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son frecuentemente implementadas incorrectamente, permitiendo a los atacantes comprometer contraseñas, llaves, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios.
A3 – Secuencia de comandos en sitios cruzados (XSS)	Los fallos XSS ocurren cada vez que una aplicación toma datos no confiables y los envía al navegador web sin una validación y codificación apropiada. XSS permite a los atacantes ejecutar secuencia de comandos en el navegador de la victima los cuales pueden secuestrar las sesiones de usuario, destruir sitios web, o dirigir al usuario hacia un sitio malicioso..
A4 – Referencia Directa Insegura a Objetos	Una referencia directa a objetos ocurre cuando un desarrollador expone una referencia a un objeto de implementación interno, tal como un fichero, directorio, o base de datos. Sin un chequeo de control de acceso u otra protección, los atacantes pueden manipular estas referencias para acceder datos no autorizados.
A5 – Configuración de seguridad incorrecta	Una buena seguridad requiere tener definida e implementada una configuración segura para la aplicación, marcos de trabajo, servidor de aplicación, servidor web, base de datos, y plataforma. Todas estas configuraciones deben ser definidas, implementadas, y mantenidas ya que por lo general no son seguras por defecto. Esto incluye mantener todo el software actualizado, incluidas las librerías de código utilizadas por la aplicación.

Figura 2.2.1. – OWASP Top 10 Vulnerabilidades (1/5)

Amenaza	Descripción
A6 – Exposición de datos sensibles	Muchas aplicaciones web no protegen adecuadamente datos sensibles tales como números de tarjetas de crédito o credenciales de autenticación. Los atacantes pueden robar o modificar tales datos para llevar a cabo fraudes, robos de identidad u otros delitos. Los datos sensibles requieren de métodos de protección adicionales tales como el cifrado de datos, así como también de precauciones especiales en un intercambio de datos con el navegador.
A7 – Ausencia de Control de Acceso a Funciones	La mayoría de aplicaciones web verifican los derechos de acceso a nivel de función antes de hacer visible en la misma interfaz de usuario. A pesar de esto, las aplicaciones necesitan verificar el control de acceso en el servidor cuando se accede a cada función. Si las solicitudes de acceso no se verifican, los atacantes podrán realizar peticiones sin la autorización apropiada.
A8 – Falsificación de Petición en Sitios Cruzados (CSRF)	Un ataque CSRF obliga al navegador de una victima autenticada a enviar una petición HTTP falsificado, incluyendo la sesión del usuario y cualquier otra información de autenticación incluida automáticamente, a una aplicación web vulnerable. Esto permite al atacante forzar al navegador de la victima para generar pedidos que la aplicación vulnerable piensa son peticiones legítimas provenientes de la victima.
A9 – Utilización de componentes con vulnerabilidades conocidas	Algunos componentes tales como las librerías, los frameworks y otros módulos de software casi siempre funcionan con todos los privilegios. Si se ataca un componente vulnerable esto podría facilitar la intrusión en el servidor o una pérdida seria de datos. Las aplicaciones que utilicen componentes con vulnerabilidades conocidas debilitan las defensas de la aplicación y permiten ampliar el rango de posibles ataques e impactos.
A10 – Redirecciones y Reenvíos no validados	Las aplicaciones web frecuentemente redirigen y reenvían a los usuarios hacia otras páginas o sitios web, y utilizan datos no confiables para determinar la página de destino. Sin una validación apropiada, los atacantes pueden redirigir a las víctimas hacia sitios de *phishing* o malware, o utilizar reenvíos para acceder páginas no autorizadas.

Figura 2.2.2. – OWASP Top 10 Vulnerabilidades (6/10)

Capítulo 3: Metodología de evaluación de seguridad

Una **metodología** [8] hace referencia al "*conjunto de procedimientos racionales o camino utilizado para alcanzar una gama de objetivos que rige una investigación, una exposición o tareas que requieran conocimientos, habilidades o cuidados específicos*".

La decisión de optar por una u otra metodología es siempre difícil, y dependerá del ámbito, pericia y propósito de la evaluación. Cada metodología aporta diferentes aspectos que serán convenientes en determinados casos, por lo que no es una decisión sencilla, y debe ser estudiada y adaptada para cada uno de estos.

Es importante remarcar que, al menos en este entorno, no hay una metodología mejor o peor que otra, sino que cada una aporta distintos puntos de vista de cara a la evaluación de seguridad y sirven como manual de referencia a la hora de realizar esta tarea.

El principal motivo de utilizar **OWASP**, como metodología de referencia a la hora de llevar a cabo esta auditoría, radica en la documentación que ofrece, la cual trata de una forma más específica y directa el modo de llevar a cabo auditorías en entornos y aplicaciones web.

En el apartado siguiente se detallarán las características y los aspectos más importantes de esta metodología.

3.1. OWASP

OWASP [9] es un proyecto de código abierto instaurado en 2001, apoyado y gestionado actualmente por la *Fundación OWASP*, organización sin ánimo de lucro constituida en 2004 y cuyo propósito es garantizar la continuidad de dicho proyecto, la cual lucha contra la causa de software inseguro, concretamente la construcción de aplicaciones y servicios más fiables, pero dedicada de forma exclusiva a entornos y aplicaciones web.

La comunidad OWASP [10], formada por organizaciones educativas, particulares y compañías de todo el mundo, trabaja para crear metodologías, documentación, artículos, tecnologías y herramientas que se liberan y pueden ser utilizadas de forma gratuita por cualquiera. Se halla libre de presiones corporativas (aunque sí asiste el uso instruido de tecnologías de seguridad comercial), lo que facilita a este nuevo tipo de entidad proporcionar información imparcial, práctica y periódicamente actualizable sobre seguridad informática.

La metodología OWASP [9] se considera muy didáctica y precisa, su exposición es muy detallada en el ámbito de la seguridad web. Hay que considerarla como un marco de

referencia que abarca tanto técnicas, como tareas que es aconsejable realizar en diversas fases del ciclo de vida del desarrollo de software (SDLC). A pesar de ser una metodología sumamente técnica, aporta una alta usabilidad gracias a una descripción muy resumida de las pruebas y a las aportaciones de ejemplos y herramientas referenciadas. Esta metodología presenta también una valoración de riesgos desde un punto de vista teórico, identificando la probabilidad de ocurrencia de cada uno de ellos y el impacto que puedan originar.

La cobertura de las pruebas, como ya se ha indicado, se centra exclusivamente en entornos y aplicaciones web, de manera que, para cubrir el espectro de las redes de infraestructura y los sistemas de soporte, necesitaríamos complementarnos con otra metodología afín.

La **Guía de pruebas OWASP 2008 v3.0**, en la que nos apoyaremos para la realización de esta auditoría, contempla y examina en detalle las pruebas que hay que estudiar para los siguientes **grupos de vulnerabilidades**:

- Recolección de información.
- Pruebas de gestión de la configuración.
- Pruebas de autenticación.
- Pruebas de gestión de sesiones.
- Pruebas de autorización.
- Comprobación de la lógica de negocio.
- Pruebas de validación de datos.
- Pruebas de denegación de servicio (DoS).
- Pruebas de servicios web.
- Pruebas de AJAX.

En el próximo apartado, detallaremos la planificación y análisis de las vulnerabilidades que posteriormente evaluaremos sobre el WCMS descrito con anterioridad, en concreto las representadas en el documento *OWASP Top 10 – 2013*.

3.2. Planificación y análisis de vulnerabilidades

El proyecto **OWASP Top 10** se centra en los riesgos, proveyendo orientación sobre cómo verificar si se poseen problemas en esta área y cómo evitarlos o evaluarlos.

Los atacantes informáticos pueden emplear potencialmente diferentes rutas alternativas para damnificar un negocio, representando cada una de ellas un riesgo que puede ser, o no, suficientemente serio como para merecer atención.

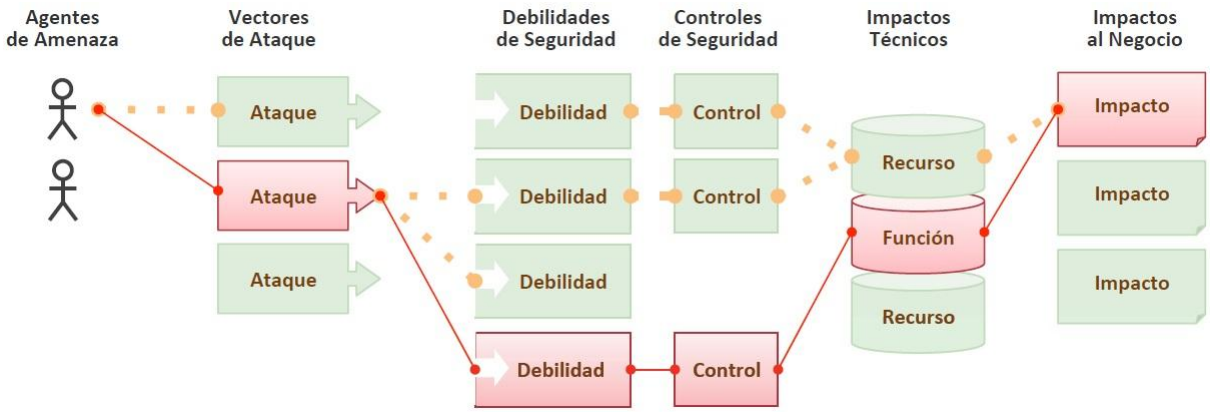

Figura 3.2.1. – Rutas de riesgo potencial para un atacante informático

Unas veces estas rutas son triviales de detectar y explotar, otras en cambio son extremadamente difíciles. De forma similar, el daño que originan puede no causar ninguna consecuencia o, en el peor de los casos, declinar el negocio.

Cada ítem en este Top 10 describe los factores de consecuencia, o impacto técnico, y la probabilidad general que se utilizan para clasificar la gravedad típica del riesgo total, empleando el siguiente esquema de calificación basado en la *Metodología de Evaluación de Riesgos de OWASP*:

Agente de Amenaza	Vectores de Ataque	Prevalencia de Debilidades	Detectabilidad de Debilidades	Impacto Técnico	Impacto al Negocio
Específico de la aplicacion	Fácil	Difundido	Fácil	Severo	Específico de la aplicación /negocio
	Promedio	Común	Promedio	Moderado	
	Difícil	Poco Común	Difícil	Menor	

Figura 3.2.2. – Esquema de calificación de gravedad de riesgos

Para una aplicación concreta, podría no existir un agente de amenaza que emprenda el ataque en cuestión, o el impacto técnico podría ser irrelevante. Por lo tanto, debemos evaluar cada riesgo dirigiéndolo hacia los agentes de amenaza, los controles de seguridad y los impactos de negocio. La denominación de los riesgos en el OWASP Top 10 surge en función del tipo de ataque, su debilidad o el impacto que provocan.

A continuación, nos centraremos en cada una de estas diez vulnerabilidades web. En particular, se dará respuesta a las siguientes cuestiones: **¿Soy vulnerable? ¿Cómo prevenirse?**

A1 – Inyección

La mejor forma de deducir si un entorno web es vulnerable a una inyección es verificar que en todo uso de intérpretes se desvincula la información confidencial del comando o consulta. Para llamadas SQL, supone servirse de variables parametrizadas, métodos almacenados y eludir consultas dinámicas. **Revisar el código fuente** es una forma efectiva de ver si un entorno web maneja intérpretes de forma fiable. Los **escaneos dinámicos automatizados** pueden proveer también una idea de si existe alguna inyección explotable, aunque no siempre se llega a estos intérpretes, perdiendo efectividad en la detección del ataque.

Se pueden destacar **dos alternativas seguras** para prevenir un ataque de inyección:

1. Servirse de una API fiable que evite el uso del intérprete íntegramente o proporcione una interfaz parametrizada.
2. Si no encontramos disponible una API parametrizada, se deben codificar a conciencia los caracteres especiales, utilizando una sintaxis de escape definida para dicho intérprete.

Figura 3.2.3. – Calificación de gravedad del riesgo A1

A2 – Pérdida de autenticación y gestión de sesiones

Los principales activos a preservar en este tipo de vulnerabilidad son las **credenciales** y los **identificadores de sesión (ID)**. Por lo tanto, podemos ser vulnerables si:

- Las credenciales se adivinan/sobrescriben por medio de funciones débiles de gestión de la sesión.
- Las credenciales no se respaldan cuando se almacenan utilizando un cifrado o hash.
- En las sesiones de usuario, los tokens de autenticación o ID de sesión no expiran.
- Los ID de sesión se muestran explícitamente en la URL o son sensibles a ataques de fijación de la sesión.
- Las credenciales y/o los ID de sesión se envían por medio de canales no cifrados (sin conexión SSL/TLS).
- Los ID de sesión no se rotan tras una correcta autenticación.

Las **dos recomendaciones** convenientes para una buena prevención ante esta vulnerabilidad son:

1. Eludir categóricamente vulnerabilidades de XSS que podrían emplearse para expoliar identificadores de sesión. *(Ver punto A3)*
2. Favorecer al programador con un único y robusto conjunto de gestión de sesiones y controles de autenticación.

Figura 3.2.4. – Calificación de gravedad del riesgo A2

A3 – Secuencia de comandos en sitios cruzados (XSS)

La siguiente vulnerabilidad que se presenta requiere cerciorarse de que todos los datos de entrada introducidos por un usuario sean fiables y se escapen adecuadamente antes de insertarse en la página de salida. Una **adecuada codificación/validación de salida** garantiza que los datos de entrada se traten siempre como texto en el navegador, en vez de contenido activo que pueda ser ejecutado.

Existen herramientas que pueden identificar de forma automática diversas vulnerabilidades de XSS, en cambio, el uso de diferentes intérpretes en el navegador dificulta considerablemente esta detección. Tecnologías como AJAX hacen que esta vulnerabilidad sea mucho más costosa de detectar mediante las herramientas automatizadas existentes. Por lo tanto, una cobertura íntegra requiere de una composición de técnicas, como las pruebas manuales de intrusión y revisión de código fuente, además de cualquier testeo automatizado.

Prevenir XSS advierte principalmente mantener los datos no confiables liberados del contenido activo del navegador. A tal objeto se exponen **tres alternativas**:

1. Implementar una validación de entradas positiva aun sin ser esta una defensa completa, ya que muchos entornos exigen aceptar caracteres especiales en sus entradas. En la medida de lo posible, dicha validación debe aprobar el largo, el formato, los caracteres y cualquier regla de negocio sobre estos datos antes de permitir su entrada.
2. Escapar los datos no confiables basados en el contexto HTML donde serán emplazados (cuerpo, atributo, JavaScript, CSS o URL).
3. Emplear políticas de seguridad de contenido (CSP) para proteger la integridad de su sitio contra este tipo de vulnerabilidad.

Figura 3.2.5. – Calificación de gravedad del riesgo A3

A4 – Referencia directa insegura a objetos

La forma más destacada de verificar si un entorno es vulnerable a referencias inseguras a objetos es **comprobar que todas las referencias a objetos poseen la seguridad adecuada**. Para ello se debe:

- Comprobar si el usuario tiene autorización de acceso al recurso específico que demanda, para referencias directas a recursos restringidos.
- Limitar la correlación con la referencia directa a valores autorizados para el usuario en particular, para referencias indirectas a recursos restringidos.

Con frecuencia, un testeo automático no revela este tipo de vulnerabilidad ya que no es capaz de identificar qué referencia a objeto necesita protección o si es segura/insegura.

Prevenir esta vulnerabilidad requiere distinguir una manera de preservar los objetos accesibles por cada usuario. **Dos formas** destacadas son:

1. Verificando el acceso para cada referencia directa a un objeto no confiable, garantizando así que el usuario está autorizado a acceder al objeto demandado.
2. Manejando referencias indirectas por sesión o usuario, obviando así que los atacantes accedan de forma directa a recursos no autorizados.

Figura 3.2.6. – Calificación de gravedad del riesgo A4

A5 – Configuración de seguridad incorrecta

Conviene comprobar si en todos los niveles de la pila del entorno web hemos fortalecido la seguridad. Para ello debemos responder a las siguientes cuestiones:

- ¿Existe software sin actualizar (OS, servidores, librerías de código, DBMS, etc.)?
- ¿Todo lo innecesario permanece deshabilitado, desinstalado o eliminado (servicios, puertos, privilegios, etc.)?
- ¿Continúan las cuentas predeterminadas aún habilitadas y sus credenciales sin modificar?
- ¿La configuración del sistema gestor de errores elude el acceso de forma no autorizada a los mensajes de error?
- ¿Las configuraciones de seguridad en las librerías y en el framework de desarrollo permanecen a valores seguros?

Sin un procedimiento de configuración de seguridad replicable y concertado para los entornos y aplicaciones web, los sistemas permanecerán en alto riesgo.

Las **recomendaciones principales** se centran en establecer:

1. Un proceso de mantenimiento que despliegue todos los parches y actualizaciones de software de manera conveniente.
2. Un proceso automatizado y repetible que posibilite configurar, fácil y rápidamente, entornos asegurados.
3. Exploraciones periódicas que asistan la detección de parches omitidos o errores de configuración.
4. Una robusta arquitectura de aplicación que otorgue una separación fiable y efectiva entre sus componentes.

Figura 3.2.7. – Calificación de gravedad del riesgo A5

A6 – Exposición de datos sensibles

En primer lugar, se debe **precisar el conjunto de datos sensibles** que adviertan una protección extra (contraseñas, información bancaria, registros médicos, etc.). Para estos datos:

- ¿Se transfieren en texto claro, externa o internamente?
- ¿Se memorizan en texto claro a largo plazo, incluyendo sus respaldos?
- ¿Se emplea algún método criptográfico antiguo o débil?
- ¿Se originan claves criptográficas débiles o falta una adecuada gestión de éstas?
- ¿Se manejan cabezales y directivas de seguridad del navegador cuando se envían/proveen datos sensibles por/para el mismo?

Las **principales sugerencias** se sitúan en establecer lo siguiente:

1. No almacenar datos susceptibles innecesariamente, datos que no se poseen no pueden ser usurpados.
2. Valorar las amenazas de las cuales proteger los datos sensibles y garantizar el cifrado de los mismos (en tráfico o almacenados).
3. Aplicar algoritmos de cifrado estandarizados y robustos, así como claves criptográficas fuertes gestionadas de forma fiable.
4. Deshabilitar el autocompletado en formularios que recopilan datos susceptibles, así como el cacheo de páginas que comprendan datos de este tipo.
5. Almacenar las claves criptográficas con un algoritmo diseñado específicamente para protegerlas.

Figura 3.2.8. – Calificación de gravedad del riesgo A6

A7 – Ausencia de control de acceso a funciones

La vulnerabilidad que se presenta a continuación requiere comprobar cada funcionalidad del entorno o aplicación web de la siguiente forma:

- ¿Existe autenticación del lado del servidor, o se han descuidado las verificaciones de autorización?
- ¿La interfaz de usuario (UI) guía la navegación hacia funcionalidades no autorizadas?
- ¿Las comprobaciones del lado del servidor se orientan de forma exclusiva a la información suministrada por el atacante?

Las herramientas automatizadas no suelen detectar este tipo de vulnerabilidad. Sí existen algunas pruebas, utilizando un **servidor proxy**, que apoyan directamente este tipo de análisis. Al mismo objeto, también se puede **examinar la implementación del control de acceso en el código fuente**, siguiendo una solicitud unitaria con privilegios y comprobando el patrón de autorización, a fin de identificar dónde no se está siguiendo éste mismo patrón.

Se debe disponer de un patrón de autorización estable y fácil de analizar. Esta defensa frecuentemente se surte de componentes externos. Las **principales recomendaciones** están encaminadas en establecer lo siguiente:

1. Por defecto, la implementación de este módulo debería impedir todo acceso, requiriendo la habilitación explícita de permisos a roles específicos para acceder a cada funcionalidad.
2. El gestor de permisos y accesos debe ser fácilmente actualizable, manejando parametrizaciones en su implementación directa sobre el código al uso.
3. Si la funcionalidad forma parte de un flujo de trabajo, se debe verificar y garantizar que las condiciones de éste se hallen en el estado oportuno para autorizar el acceso.

Figura 3.2.9. – Calificación de gravedad del riesgo A7

A8 – Falsificación de peticiones en sitios cruzados (CSRF)

La manera más sencilla de identificar una vulnerabilidad en un entorno o aplicación web es **comprobar la omisión de un testigo (*token*) impredecible en cada formulario y enlace**. Si no se identifica dicho testigo los atacantes pueden falsificar peticiones, por lo que debemos concentrar el análisis en formularios y enlaces que invoquen funciones que posibiliten cambiar estados (por ejemplo, un CAPTCHA).

Se deben constatar las operaciones que impliquen múltiples pasos, ya que no están exentos a este tipo de ataque. De igual forma, descartar como protección las direcciones IP de origen, las cookies de sesión y otro tipo de información, ya que ésta se encuentra también incluida en las peticiones falsificadas. Existe una herramienta – CSRF Tester – elaborada por OWASP, que nos ayuda a detectar fallos relacionados con este tipo de vulnerabilidad.

Para prevenir la CSRF se sugiere la inclusión de un testigo impredecible en el cuerpo, o URL, de cada petición HTTP. Como mínimo, dicho testigo debe ser único por cada sesión de usuario. A continuación se presentan **dos alternativas**:

1. Preferiblemente se debe insertar el testigo en un campo oculto, generando que el valor se envíe en el cuerpo de la petición HTTP, evitando así su inclusión y exposición en la URL.
2. Advertir que el usuario vuelva a autenticarse, o desarrollar una validación que verifique que se trata de un usuario legítimo, puede protegernos frente a ataques de este tipo.

Figura 3.2.10. – Calificación de gravedad del riesgo A8

A9 – Utilización de componentes con vulnerabilidades conocidas

Teóricamente debiera no ser difícil diferenciar si se está utilizando un componente vulnerable, pero de manera desventurada los informes de vulnerabilidades para software de código abierto o comercial no siempre establecen de forma accesible qué versión de un componente es vulnerable dentro un estándar.

Por lo tanto, para decretar si un componente es vulnerable debemos cachear en bases de datos, listas de correos de los proyectos y anuncios de vulnerabilidades. Del mismo modo, si un componente posee una vulnerabilidad, se debe revisar minuciosamente la parte del código fuente que utiliza a este componente concreto, comprobando si este error puede derivar en un impacto del cual cuidarnos.

Prevenir este tipo de vulnerabilidad reclama una serie de **recomendaciones fundamentales**:

1. Revisar la fiabilidad de los componentes y mantenerlos actualizados.
2. Fichar todos los componentes y la versión en uso, incluyendo sus dependencias.
3. Vincular capas de seguridad entorno al componente para inhabilitar funcionalidades no utilizadas y/o asegurar las trazas vulnerables de este.
4. Decretar políticas de seguridad que regularicen el manejo de componentes.

Figura 3.2.11. – Calificación de gravedad del riesgo A9

A10 – Redirecciones y reenvíos no validados

La mejor manera de verificar si un entorno o aplicación web dispone de reenvíos y redirecciones no validadas es:

- Analizar el código fuente para determinar si la URL de destino se encuadra en el valor de algún parámetro.
- Examinar los parámetros facilitados antes de la redirección para comprobar si son una URL objetivo o un recurso de ésta.
- Recorrer la aplicación percibiendo si genera redirecciones (réplicas HTTP 300-307, popularmente 302).
- Si no se dispone del código fuente, debemos examinar todos los parámetros y comprobar si forman parte de un reenvío o redirección de una URL objetivo.

Puede llevarse a cabo un **manejo seguro de redirecciones y reenvíos** de varias formas:

1. Si se emplean, no comprometer parámetros manipulables por el usuario para determinar el objetivo final.
2. Si los parámetros de destino no pueden eludirse, garantizar que el valor proporcionado es lícito y autorizado para el usuario. Es aconsejable que el valor de cualquier parámetro de destino sea un valor de mapeo en vez de la URL destino, a fin de realizar la traducción pertinente en el lado del servidor.
3. Requerir SSL/TLS para todas las páginas sensibles. Las peticiones sin SSL/TLS a estas páginas se deben redirigir a las correspondientes páginas con SSL/TLS.
4. Configurar el servidor SSL/TLS para aceptar solo algoritmos de nivel fuerte.
5. Implementar con el atributo "*secure*" todas las cookies sensibles.
6. Las conexiones a sistemas *back-end* deben emplear tecnología SSL/TLS.
7. Comprobar que los certificados sean válidos y se adapten a todos los dominios.

Figura 3.2.12. – Calificación de gravedad del riesgo A10

3.3. Fase de actuación: testeo y batería de pruebas

La fase de actuación ideada para esta auditoría de seguridad se llevará a cabo aplicando la siguiente táctica o procedimiento:

- Por un lado, se empleará la **herramienta automatizada OWASP ZAP v2.4.2**, la cual nos permitirá de una manera cómoda y sencilla estudiar a fondo la mayor parte de nuestro WCMS, ofreciéndonos una estimación real de las posibles amenazas.

- Por otro lado, como se ha indicado en apartados anteriores, existen puntos críticos que ninguna herramienta automatizada hasta la fecha puede alcanzar y nos veremos obligados a realizar determinadas tareas "a mano", es decir, revisar el WCMS **a nivel de código fuente y sobre la propia UI de este** en busca de esas posibles vulnerabilidades, siguiendo principalmente las pautas recomendadas en la **Guía de pruebas OWASP 2008 v3.0**.

Con todo ello, se pretenderá reducir en la medida de lo posible el mayor número real de puntos potencialmente vulnerables, asumiendo aun así la inexistencia de un software plenamente seguro.

A continuación, se presenta la herramienta automatizada con la que se efectuará un importante y principal chequeo del WCMS:

OWASP ZAP [11] es una potente herramienta multiplataforma (libre y gratuita), especializada en ataques de penetración que posibilita analizar entornos web y registrar sus vulnerabilidades con el fin principal de securizar dichos entornos. Para su desarrollo, ZAP se apoya en la lista de las vulnerabilidades web más usuales de OWASP, siendo capaz de detectar prácticamente cualquiera de estas, ofrecer a su vez la posibilidad de reportar informes y vincularse con otras aplicaciones a objeto de potenciar su análisis. Algunas de las **funcionalidades más destacadas y análisis específicos** de esta *herramienta forense de seguridad* son:

- Posibilita la asignación de un sistema de prioridades.
- Posibilita identificar recursos en un servidor.
- Posibilita la verificación de todas las solicitudes/respuestas entre cliente-servidor.

- Posibilita ejecutar ataques múltiples de inyección.
- Análisis automáticos, pasivos y de sistemas de autenticación.
- Disposición de uso de certificados SSL dinámicos o personales.
- Elección de reglas de análisis dentro del sistema de políticas de ZAP.
- Búsqueda de vulnerabilidades en “modo ataque”.
- Notifica recomendaciones para la reparación de vulnerabilidades.

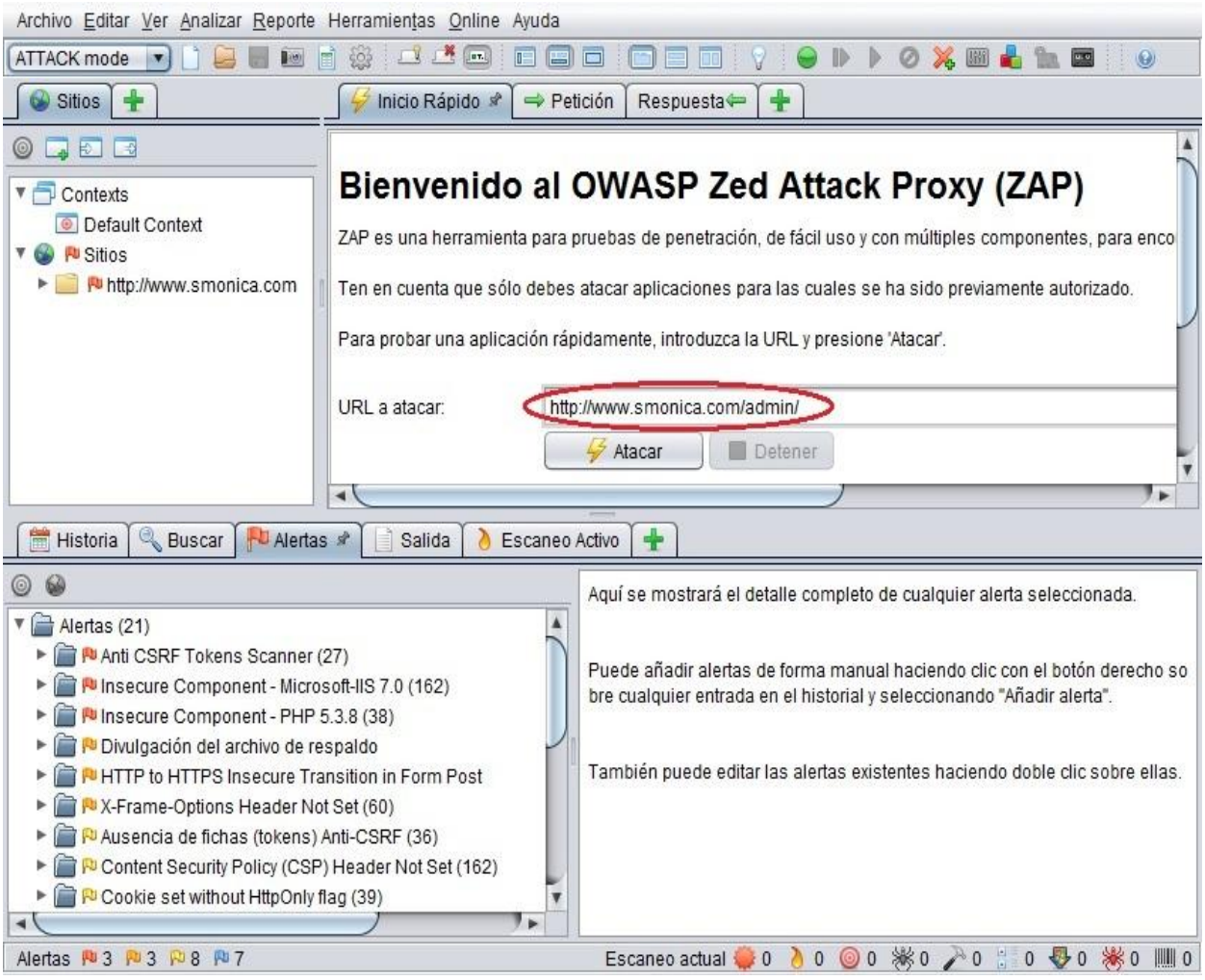

Figura 3.3.1. – Interfaz de usuario de la herramienta OWASP ZAP v2.4.2

Si tuviésemos que enmarcar la fase de actuación de esta auditoría, dentro de una de las tres categorías de test de intrusión existentes (caja negra – caja blanca – caja gris), a la hora de comprobar la seguridad de este WCMS, nos competería la categoría de **caja gris**:

Un **test de caja gris** [12] lleva a cabo pruebas con métodos semejantes a los de caja negra, imitando ataques reales, aunque, a diferencia de estos, el atacante posee información técnica concerniente al sistema en cuestión y, asimismo, se le posibilita solicitar información adicional marcada con comentarios, de igual forma que en los test de caja blanca.

Seleccionar este tipo de test nos garantiza una alta rentabilidad en tiempo y una efectividad constatada a la hora de proporcionar el mayor número real de vulnerabilidades y de perpetrar estimaciones de seguridad cuando los valores más absolutos de las cajas negra y blanca no son imprescindibles.

En el capítulo que sucede, se expondrán de forma detallada los resultados pertinentes a esta fase de actuación, con el planteamiento de los problemas descubiertos, junto a los datos relevantes y las recomendaciones consumadas para la subsanación de las vulnerabilidades detectadas.

Capítulo 4: Resultados

En esta sección se muestran los resultados de la auditoría llevada a cabo sobre el WCMS descrito anteriormente. En primer lugar, la Tabla 4.1 ofrece un resumen con las vulnerabilidades detectadas, dentro del marco Top 10 de OWASP, junto a los datos más significativos de cada problema[1].

AMENAZA DETECTADA	DATOS RELEVANTES
A2 – Pérdida de autenticación y gestión de sesiones	• Conjunto de cookies sin indicador *"HttpOnly"* • Detección de cookies inactivas
A3 – Secuencia de comandos en sitios cruzados (XSS)	• CSP no establecida en cabeceras HTTP • Protección XSS no habilitada para el navegador
A5 – Configuración de seguridad incorrecta	• Omisión del campo "*X-Frame-Options*" en cabeceras *HTTP response* • Filtración de información a través de los campos "*X-Powered-By*" y "*Server*" en cabeceras *HTTP response* • Configuración susceptible del campo "*X-Content-Type-Options*" en cabeceras *Anti-MIME-Sniffing*
A6 – Exposición de datos sensibles	• Divulgación de archivos de respaldo • Autocompletado de campos no confiables en el navegador • Aplicación liberal de cookies • Contenido (no)almacenable y (no)cacheable
A8 – Falsificación de peticiones en sitios cruzados (CSRF)	• Escaneo de tokens anti-CSRF • Omisión de tokens anti-CSRF
A9 – Utilización de componentes con vulnerabilidades conocidas	• Componente desactualizado – Microsoft IIS 7.0 • Componente desactualizado – PHP 5.3.8
A10 – Redirecciones y reenvíos no validados	• Requerimiento de HTTPS en secciones comprometidas

Tabla 4.1. – Cuadro resumen de resultados con datos relevantes

[1] *Se ha de tener en cuenta, que la omisión de cualquiera de las amenazas del Top 10 durante los capítulos 4 y 5 del presente trabajo, manifestará de manera sobreentendida la inexistencia de dicha(s) vulnerabilidad(es) sobre el WCMS auditado.*

4.1. Descripción de resultados obtenidos

A continuación se describirán de forma objetiva y explícita, sin valoraciones ni justificaciones, los resultados obtenidos para cada vulnerabilidad detectada.

A2 – Pérdida de autenticación y gestión de sesiones

PROBLEMA	RIESGO	DESCRIPCIÓN
Conjunto de cookies sin indicador "*HttpOnly*"	BAJO	Una cookie se ha establecido sin indicador "*HttpOnly*", lo que significa que puede ser accedida a través de un script malicioso.
Detección de cookies inactivas	INFORMATIVO	Repetición de solicitudes GET: colocar una cookie diferente cada vez, junto a una *request* con todas las cookies, para estabilizar la sesión, compara las *responses* contra la línea base GET original.

Tabla 4.1.1. – Descripción de resultados para la vulnerabilidad A2

A3 – Secuencia de comandos en sitios cruzados (XSS)

PROBLEMA	RIESGO	DESCRIPCIÓN
CSP no establecida en cabeceras HTTP	BAJO	CSP (Content Security Policy) es una capa adicional de seguridad que ayuda a detectar y mitigar diversos tipos de ataques, incluyendo XSS (Cross-Site Scripting) y ataques de inyección de datos.
Protección XSS no habilitada para el navegador	BAJO	La protección XSS no está habilitada para el navegador web, o está desactivada desde el servidor la configuración "*X-XSS-Protection*" en las cabeceras *HTTP response*.

Tabla 4.1.2. – Descripción de resultados para la vulnerabilidad A3

A5 – Configuración de seguridad incorrecta

PROBLEMA	RIESGO	DESCRIPCIÓN
Omisión del campo "*X-Frame-Options*" en cabeceras *HTTP response*	MEDIO	El campo "*X-Frame-Options*" no está incluido en las cabeceras *HTTP response* para protegerse contra ataques de '*Clickjacking'*.
Filtración de información a través de los campos "*X-Powered-By*" y "*Server*" en cabeceras *HTTP response*	BAJO	El servidor web está filtrando información a través de una o más cabeceras *HTTP response*, en los campos "*X-Powered-By*" y "*Server*".
Configuración susceptible del campo "*X-Content-Type-Options*" en cabeceras *Anti-MIME-Sniffing*	BAJO	El campo "*X-Content-Type-Options*" en las cabeceras *Anti-MIME-Sniffing* no está establecido como '*nosniff*'.

Tabla 4.1.3. – Descripción de resultados para la vulnerabilidad A5

A6 – Exposición de datos sensibles

PROBLEMA	RIESGO	DESCRIPCIÓN
Divulgación de archivos de respaldo	MEDIO	Una copia de seguridad de un archivo está siendo divulgada por el servidor web.
Autocompletado de campos no confiables en el navegador	BAJO	El atributo '*AUTOCOMPLETE*' no está desactivado en el HTML (*form/input*), para un elemento de entrada de tipo '*password*'.
Aplicación liberal de cookies	INFORMATIVO	Las cookies se pueden aplicar por dominio o por ruta. Esta comprobación concierne sólo al ámbito de dominio. La aplicación liberada de cookies es común en mega-aplicaciones como *google.com* y *live.com.*
Contenido no almacenable	INFORMATIVO	Los contenidos de respuesta no son almacenables por los componentes de almacenamiento en caché, como los servidores *proxy*.
Contenido almacenable y cacheable	INFORMATIVO	Los contenidos de respuesta son almacenables por los componentes de almacenamiento en caché, como los servidores *proxy*. Estos pueden ser recuperados directamente de la memoria caché, en lugar de desde el servidor de origen, por los servidores de almacenamiento en caché, en respuesta a solicitudes similares de otros usuarios.

Tabla 4.1.4. – Descripción de resultados para la vulnerabilidad A6

A8 – Falsificación de peticiones en sitios cruzados (CSRF)

PROBLEMA	RIESGO	DESCRIPCIÓN
Escaneo de tokens anti-CSRF	ALTO	Una falsificación de petición en sitios cruzados (CSRF) es un ataque que consiste en obligar a la víctima a enviar un *HTTP request* a un objetivo, sin su conocimiento o intención, con el fin de ejecutar una acción suplantando la identidad de esta víctima.
Omisión de tokens anti-CSRF	BAJO	No se encontraron tokens anti-CSRF en formularios HTML.

Tabla 4.1.5. – Descripción de resultados para la vulnerabilidad A8

A9 – Utilización de componentes con vulnerabilidades conocidas

PROBLEMA	RIESGO	DESCRIPCIÓN
Componente desactualizado – Microsoft IIS 7.0	ALTO	Basado en la respuesta del análisis pasivo, se muestra como componente en uso inseguro: Microsoft IIS 7.0
Componente desactualizado – PHP 5.3.8	ALTO	Basado en la respuesta del análisis pasivo, se muestra como componente en uso inseguro: PHP 5.3.8

Tabla 4.1.6. – Descripción de resultados para la vulnerabilidad A9

A10 – Redirecciones y reenvíos no validados

PROBLEMA	RIESGO	DESCRIPCIÓN
Requerimiento de HTTPS en secciones comprometidas	MEDIO	El protocolo de aplicación HTTPS emplea un canal cifrado más apropiado, para el tráfico de información sensible, que el protocolo HTTP.

Tabla 4.1.7. – Descripción de resultados para la vulnerabilidad A10

4.2. Análisis e interpretación de resultados

En el presente apartado, se llevará a cabo el estudio de cada uno de los resultados descritos en el punto 4.1., a fin de adquirir una comprensión más transparente sobre los mismos.

A2 – Pérdida de autenticación y gestión de sesiones

PROBLEMA	URLs AFECTADAS	ESTUDIO
Conjunto de cookies sin indicador "*HttpOnly*"	http://www.smonica.com/admin/ …/admin/modulos/macceso.php	Si un script malicioso se ejecuta en código, las cookies afectadas a este fallo serán accesibles y podrán ser transferidas a otros sitios. Si se trata de una cookie de sesión, ésta puede llegar a ser secuestrada.
Detección de cookies inactivas	http://www.smonica.com/admin/ …/admin/modulos/macceso.php	Se pueden revelar áreas donde las cookies basadas en autenticación y atributos no se están aplicando realmente. Las cookies que no tienen efectos esperados pueden revelar defectos en la lógica de la aplicación. En el peor de los casos, este fallo puede revelar dónde no se aplican realmente cookies basadas en autenticación por token(s).

Tabla 4.2.1. – Estudio de resultados para la vulnerabilidad A2

A3 – Secuencia de comandos en sitios cruzados (XSS)

PROBLEMA	URLs AFECTADAS	ESTUDIO
CSP no establecida en cabeceras HTTP	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	XSS se utiliza para muchos fines maliciosos, desde el robo de datos a la distribución de malware. CSP proporciona un conjunto de cabeceras HTTP estándar que permiten a los propietarios de sitios web declarar fuentes de contenido aprobadas, las cuales los navegadores deben autorizar a la hora de cargarse sobre las páginas del sitio en cuestión. JavaScript, CSS, *HTML frames*, fuentes, imágenes y objetos embebidos, como *applets* de Java, *ActiveX*, archivos de audio y vídeo, son tipos de fuentes de contenido que deben cubrirse por medio de una CSP.

Protección XSS no habilitada para el navegador	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	El campo "*X-XSS-Protection*", en las cabeceras *HTTP response*, permite al servidor web (des)activar el mecanismo de protección XSS del navegador. Los siguientes valores, respectivamente, activarían y desactivarían esta protección: - X-XSS-Protection: 1 - X-XSS-Protection: 0 Actualmente esta protección está soportada en los siguientes navegadores: Internet Explorer - Chrome - Safari (KebKit)

Tabla 4.2.2. – Estudio de resultados para la vulnerabilidad A3

A5 – Configuración de seguridad incorrecta

PROBLEMA	URLs AFECTADAS	ESTUDIO
Omisión del campo "*X-Frame-Options*" en cabeceras *HTTP response*	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	"*X-Frame-Options*" es un encabezado HTTP que debe configurarse directamente en el servidor para negar/permitir que otros dominios puedan visualizar nuestro sitio web dentro de *iFrames*.
Filtración de información a través de los campos "*X-Powered-By*" y "*Server*" en cabeceras *HTTP response*	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	El acceso a la información que se filtra a través de estos encabezados, puede facilitar a los atacantes la identificación de frameworks y componentes de su aplicación web, incrementando así su vulnerabilidad.

Configuración susceptible del campo "*X-Content-Type-Options*" en cabeceras *Anti-MIME-Sniffing*	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	Este error permite que antiguas versiones de Internet Explorer y Google Chrome ejecuten '*MIME-sniffing'* en el cuerpo de la respuesta, ocasionando que ésta se interprete y se muestre como un tipo de contenido no declarado. Las versiones actuales y oficiales de Firefox utilizan (si así se establece) tipos de contenido declarado, en lugar de realizar '*MIME-sniffing'*.

Tabla 4.2.3. – Estudio de resultados para la vulnerabilidad A5

A6 – Exposición de datos sensibles

PROBLEMA	URLs AFECTADAS	ESTUDIO
Divulgación de archivos de respaldo	…/admin/js/jquery.dd.js	Los archivos de respaldo deben almacenarse fuera de su sitio web, en distinto servidor que éste.
Autocompletado de campos no confiables en el navegador	http://www.smonica.com/admin/	El atributo '*AUTOCOMPLETE*' nos ayuda a recordar, autocompletar y/o sugerir los valores insertados con anterioridad en el mismo formulario. Debemos prestar especial atención a qué entradas de texto aplicar esta función dentro de los formularios.
Aplicación liberal de cookies	http://www.smonica.com/admin/ …/admin/modulos/macceso.php	El campo de aplicación de las cookies, por defecto, se limita a todas las direcciones URL del actual *host name*. Sin embargo, el campo de aplicación puede limitarse también a una ruta o *path*, a la cual enviar la cookie, o ampliarse a un dominio completo. En este último caso se podría especificar también otra limitación, estableciendo una cookie para cualquier segmento de su *host name*.
Contenido no almacenable	http://www.smonica.com/admin/ …/admin/modulos/macceso.php	Si la respuesta no contiene información confidencial, personal o específica del usuario, puede beneficiarnos almacenarla en caché, a fin de mejorar el rendimiento.

Contenido almacenable y cacheable	…/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	Si los datos de respuesta son personales o específicos del usuario, esto puede dar lugar a una filtración de información susceptible. En algunos casos, esto puede incluso dar lugar a que un usuario obtenga el control completo de la sesión de otro usuario, dependiendo de la configuración de los componentes de almacenamiento en caché para ese entorno. Esta es la principal causa por la que los servidores de almacenamiento en caché compartidos, como cachés *proxy*, se configuran en red local. Esta configuración se encuentra frecuentemente en entornos corporativos o educativos.

Tabla 4.2.4. – Estudio de resultados para la vulnerabilidad A6

A8 – Falsificación de peticiones en sitios cruzados (CSRF)

PROBLEMA	URLs AFECTADAS	ESTUDIO
Escaneo de tokens anti-CSRF	http://www.smonica.com/admin/ …/admin/js/jquery.js	La causa subyacente es la funcionalidad de la aplicación utilizando acciones *URL/form* predecibles de manera redundante. La naturaleza del ataque CSRF es explotar la confianza que un sitio web tiene en un usuario. Por el contrario, XSS explota la confianza que un usuario tiene en un sitio web. Los ataques CSRF son eficaces en una serie de situaciones, incluyendo: - La víctima tiene una sesión activa en el sitio objetivo. - La víctima se autentifica vía HTTP en el sitio objetivo. - La víctima se encuentra en la misma red local que el sitio objetivo. CSRF se ha utilizado para realizar acciones contra sitios objetivo utilizando los privilegios de la víctima, pero las técnicas recientemente descubiertas permiten divulgar información mediante el acceso a la respuesta. El riesgo de divulgación de información se incrementa notablemente cuando el sitio objetivo es vulnerable también a XSS, convirtiéndolo en una plataforma para CSRF, lo que permite el ataque para operar con las mismas limitaciones y políticas que en el sitio origen.

Omisión de tokens anti-CSRF	http://www.smonica.com/admin/	No se ha encontrado ningún token anti-CSRF (*anticsrf, CSRFToken, __RequestVerification Token, csrfmiddlewaretoken, authenticity_ token*) en los siguientes formularios: - HTML: [Formulario 1: "usuario" "Contrasena" "entrar"]

Tabla 4.2.5. – Estudio de resultados para la vulnerabilidad A8

A9 – Utilización de componentes con vulnerabilidades conocidas

PROBLEMA	URLs AFECTADAS	ESTUDIO
Componente desactualizado – Microsoft IIS 7.0	http://www.smonica.com/admin/ …/admin/modulos/macceso.php …/admin/css/estilos1/estilos.css …/admin/css/estilos1/formulario.css …/admin/css/estilos1/lightbox.css …/admin/css/estilos1/listas.css …/admin/js/expose.js …/admin/js/funciones.js …/admin/js/jquery.js …/admin/js/tools.expose.1.0.5.js	Existen nuevas versiones recientes y actualizadas para este componente. La utilización de versiones antiguas de componentes dentro de este ámbito incrementa potencialmente el riesgo de ataque al entorno.
Componente desactualizado – PHP 5.3.8	http://www.smonica.com/admin/ …/admin/modulos/macceso.php	Existen nuevas versiones recientes y actualizadas para este componente. La utilización de versiones antiguas de componentes dentro de este ámbito incrementa potencialmente el riesgo de ataque al entorno.

Tabla 4.2.6. – Estudio de resultados para la vulnerabilidad A9

A10 – Redirecciones y reenvíos no validados

PROBLEMA	URLs AFECTADAS	ESTUDIO
Requerimiento de HTTPS en secciones comprometidas	http://www.smonica.com/admin/	HTTPS utiliza un cifrado basado en SSL/TLS, dos protocolos que proporcionan comunicaciones con mayor nivel de seguridad en una red que el HTTP tradicional.

Tabla 4.2.7. – Estudio de resultados para la vulnerabilidad A10

4.3. Resolución

Para finalizar este capítulo, se hará mención nuevamente de las diversas vulnerabilidades detectadas en esta auditoría y propondremos las soluciones pertinentes recomendables a cada una de ellas. Todo ello se verá expuesto a través de la siguiente Tabla:

	PROBLEMA	SOLUCIÓN
A2	**Conjunto de cookies sin indicador "*HttpOnly*"**	Comprobar que el indicador "*HttpOnly*" se establece para todas las cookies.
	Detección de cookies inactivas	Revisar la política de cookies para un correcto uso e implementación de éstas: - http://politicadecookies.com/
A3	**CSP no establecida en cabeceras HTTP**	Comprobar que el servidor web/de aplicación está configurado para establecer todas las cabeceras con CSP, logrando así un apoyo óptimo hacia el navegador: - "*Content-Security-Policy*" para Chrome 25+, Firefox 23+ y Safari 7+ - "*X-Content-Security-Policy*" para Firefox 4.0+ e Internet Explorer 10+ - "*X-WebKit-CSP*" para Chrome 14+ y Safari 6+
	Protección XSS no habilitada para el navegador	Comprobar que el filtro XSS del navegador web está habilitado, estableciendo en la cabecera *HTTP response* el valor '1' para el campo "*X-XSS-Protection*".
A5	**Omisión del campo "*X-Frame-Options*" en cabeceras *HTTP response***	La mayoría de los navegadores web modernos soportan el encabezado "*X-Frame-Options*" de HTTP. Comprobar que está establecido en todas las páginas de su sitio web.
	Filtración de información a través de los campos "*X-Powered-By*" y "*Server*" en cabeceras *HTTP response*	Comprobar que su servidor web/de aplicación está configurado para suprimir encabezados "*X-Powered-By*", "*Server*" o cualquiera que proporcione detalles genéricos.
	Configuración susceptible del campo "*X-Content-Type-Options*" en cabeceras *Anti-MIME-Sniffing*	Comprobar que el servidor de aplicaciones establezca el encabezado "*Content-Type*" de forma apropiada y el de "*X-Content-Type-Options*" con valor '*nosniff*' en todas las páginas. Si es posible también, comprobar que el usuario final utilice un navegador web moderno y compatible con los estándares, que no ejecute '*MIME-sniffing*' en ninguna circunstancia, o al menos que desde el servidor web/de aplicación se pueda controlar su uso.
	Divulgación de archivos de respaldo	No editar archivos *in-situ* en el servidor web y asegurarse de que los archivos innecesarios (incluyendo archivos ocultos) se eliminan desde éste.

A6	**Autocompletado de campos no confiables en el navegador**	Deshabilitar el atributo '*AUTOCOMPLETE*' en los elementos *form/input* que contengan contraseñas utilizando: AUTOCOMPLETE='OFF'
	Aplicación liberal de cookies	Establecer siempre el ámbito de aplicación de cookies a un FQDN - Fully Qualified Domain Name -
	Contenido no almacenable	El contenido se puede marcar como almacenable al asegurarse de las siguientes condiciones: - El método de la solicitud debe ser comprensible por la caché y definido como cacheable (GET, HEAD y POST se definen actualmente como cacheables). - El código de estado de la respuesta debe ser comprensible por la caché (1XX, 2XX, 3XX, 4XX o 5XX son generalmente tipos de respuesta comprensibles). - La directiva de caché '*no-store*' no debe aparecer en los campos de las cabeceras de solicitud o respuesta. - Para el almacenamiento en cachés compartidas, como la caché *proxy*, la directiva '*private*' no debe aparecer en las cabeceras de respuesta, y la directiva '*authorization*' no debe aparecer en las cabeceras de solicitud (a menos que la respuesta lo permita explícitamente). Además de las condiciones anteriores, también se debe satisfacer en la respuesta al menos uno de los siguientes requisitos: - Se debe contener un encabezado "*Expires*". - Se debe contener una directiva de respuesta '*max-age*'. - Para cachés compartidas, como la caché *proxy*, se debe contener una directiva de respuesta '*s-maxage*'. - Se debe contener un "*Cache Control Extension*" que permita almacenamiento en caché. - Se debe tener un código de estado que se defina como cacheable por defecto (200, 203, 204, 206, 300, 301, 404, 405, 410, 414 o 501).
	Contenido almacenable y cacheable	Validar que la respuesta no contenga información confidencial, personal o específica del usuario. Si lo hace, considerar el uso de las siguientes cabeceras *HTTP response* para limitar o impedir que el contenido sea almacenado y recuperado desde la caché por otro usuario: - Cache-Control: *no-cache, no-store, must-revalidate* o *private* - Pragma: *no-cache* - Expires: 0 Así, conseguimos no almacenar respuestas ni recuperarlas (sin validación) de la caché, en respuesta a una petición similar. Esta configuración es válida tanto para HTTP 1.0 como para HTTP 1.1.

A8	**Escaneo y omisión de tokens anti-CSRF**	- Utilizar una biblioteca o framework que no permita que esta debilidad se produzca, o utilice paquetes anti-CSRF que eviten esta debilidad, como la *OWASP CSRFGuard*. - Comprobar que su aplicación está libre de riesgos XSS, ya que la mayoría de las coberturas CSRF pueden ser anuladas mediante ataques XSS. - Generar un único token para cada formulario y asegurar que no sea predecible. - Identificar las operaciones especialmente susceptibles. Cuando un usuario realice una operación de este tipo, envíe una solicitud de confirmación independiente para garantizar la fiabilidad de esa operación. - Utilizar el control ESAPI para la gestión de sesiones. Este control incluye un componente para CSRF. - No utilizar el método GET para cualquier solicitud que desencadene un cambio de estado. - Comprobar la cabecera *HTTP Referer* para verificar si la solicitud se originó a partir de una página de espera, ya que los usuarios o *proxies* pueden haber desactivado el envío del *Referer* por razones de privacidad.
A9	**Componente desactualizado – Microsoft IIS 7.0**	- Actualizar Microsoft IIS 7.0 a la última versión estable del producto. - Utilizar un gestor de paquetes, junto a sus políticas de procedimiento, para administrar las versiones instaladas de paquetes de software.
	Componente desactualizado – PHP 5.3.8	- Actualizar PHP 5.3.8 a la última versión estable del producto. - Utilizar un gestor de paquetes, junto a sus políticas de procedimiento, para administrar las versiones instaladas de paquetes de software.
A10	**Requerimiento de HTTPS en secciones comprometidas**	Dependiendo del servidor web/de aplicación y la configuración que éste utilice, se deberá emplear un método de implementación específico para este protocolo. Consultar las siguientes fuentes: - Para IIS: https://support.microsoft.com/es-es/kb/324069 - Para Apache: http://www.htmlpoint.com/apache/10.htm

Tabla 4.3.1. – Soluciones recomendadas para las vulnerabilidades detectadas

Capítulo 5: Conclusiones

En el presente Trabajo Fin de Máster se ha llevado a cabo una **auditoría de seguridad informática sobre un WCMS propiedad de la PYME española "*Soluciones Informáticas Santa Mónica S.L.*"**. En este sentido, esta sección remarca las principales conclusiones obtenidas sobre dicha auditoría.

En los siguientes apartados trataremos de sintetizar, de forma breve y concisa, el transcurso completo de este trabajo; comenzaremos resumiendo las **fases de desarrollo y análisis crítico** de la auditoría, seguido de una sección donde se hará **balance de los resultados** obtenidos frente a los que en un principio nos habíamos propuesto y, por último, reconocer las posibles **líneas futuras** que no se han podido desarrollar con mayor profundidad en este trabajo, tanto por los objetivos del mismo como por los plazos de entrega.

5.1. Resumen final

Partiendo de un **estudio previo**, tanto de la situación actual que atraviesan los WCMS y su seguridad, como del software específico y la metodología de evaluación que hemos tomado como referencia (*Guía de pruebas OWASP 2008 v3.0*) para cumplir con el propósito final de este trabajo, comienzan así a cimentarse las primeras labores de esta auditoría.

Previo a concluir este marco teórico, se realizó una **planificación** del espectro de vulnerabilidades que abarcaríamos (*OWASP Top 10 – 2013*), con el respectivo **análisis** de cada una de ellas, a fin de sustentar todo un conglomerado de conocimientos, técnicas y recursos diferenciados que nos servirían después para emprender la **fase de actuación**.

Una vez establecidos y expuestos los **objetivos generales y específicos** de esta auditoría, de los cuales hablaremos en el siguiente apartado, damos por concluido el escenario teórico del presente trabajo.

Llegamos, ahora sí, al encuadre principal de la auditoría: **el testeo y la batería de pruebas** del WCMS en cuestión. La distribución de esta fase tomó **dos vertientes** claramente diferenciadas; por un lado nos apoyamos en una **herramienta automatizada** (*OWAS ZAP v2.4.2*) con la cual lanzamos una serie de ataques de intrusión sobre este WCMS, a objeto de descubrir un amplio número de amenazas y, por otro lado, se llevó a cabo una **revisión exhaustiva "*a mano*" del código fuente y la interfaz de usuario (UI)** de dicho WCMS,

con el propósito de detectar vulnerabilidades que se escapan prácticamente del alcance de cualquier herramienta automatizada.

Detectados ya todos los problemas de seguridad, se han enmarcando posteriormente dentro de las 10 posibles vulnerabilidades fijadas como objetivo para este trabajo. A este proceso de **categorización**, le sigue una **fase de descripción y análisis** para todos los resultados obtenidos en las pruebas precedentes.

Por último, un apartado de resolución nos detalla todas y cada una de las **soluciones recomendadas**, con las cuales cumplir el objetivo prioritario de esta auditoría: **incrementar los niveles de seguridad del WCMS evaluado**.

Finalizada la descripción cronológica del proceso de desarrollo de esta auditoría, daremos paso al **análisis crítico**, donde se valorará el estado de seguridad del WCMS evaluado. Para ello, debemos disgregar principalmente las vulnerabilidades no detectadas, que presuponen una correcta implementación, de las que sí se han detectado, y que habría que solucionar.

Siguiendo esta pauta, el software auditado presenta una **defensa robusta sobre 3 vulnerabilidades**:

- ✓ A1 – Inyección
- ✓ A4 – Referencia directa insegura a objetos
- ✓ A7 – Ausencia de control de acceso a funciones

Los **puntos clave** más relevantes que se han investigado, para asentir que estas tres áreas exhiben un **nivel de seguridad satisfactorio**, son:

- La correcta codificación de caracteres a través de una sintaxis de escape definida.
- La comprobación de acceso a recursos restringidos.
- La verificación de autenticación en el lado del servidor.

Por otro lado, en cambio, se han detectado un total de **17 amenazas de seguridad** distribuidas, dentro del marco Top 10 de OWASP, de la siguiente manera:

- ✓ A2 – Pérdida de autenticación y gestión de sesiones (2)
- ✓ A3 – Secuencia de comandos en sitios cruzados - XSS (2)
- ✓ A5 – Configuración de seguridad incorrecta (3)
- ✓ A6 – Exposición de datos sensibles (5)
- ✓ A8 – Falsificación de peticiones en sitios cruzados - CSRF (2)
- ✓ A9 – Utilización de componentes con vulnerabilidades conocidas (2)
- ✓ A10 – Redirecciones y reenvíos no validados (1)

Cabe destacar, entre todas estas amenazas de seguridad, las que presentan un **riesgo de nivel medio y alto**, ya que deberían de ser prioritarias a la hora de solventarse:

RIESGO ALTO:

- Escaneo de tokens anti-CSRF.
- Componente desactualizado - Microsoft IIS 7.0
- Componente desactualizado - PHP 5.3.8

RIESGO MEDIO:

- Omisión del campo "*X-Frame-Options*" en cabeceras *HTTP response*.
- Divulgación de archivos de respaldo.
- Requerimiento de HTTPS en secciones comprometidas.

Asimismo, se debe tener muy en cuenta que, de las 17 amenazas de seguridad descubiertas, al margen de los niveles de riesgo, 6 tienen vinculación directa con las **cabeceras HTTP en el servidor** y 3 con el manejo de **cookies**, dato que nos facilita la ubicación de los dos puntos críticos más relevantes sobre los que prestar atención con mayor premura.

Tomando todas estas consideraciones, se poseen indicios suficientes para advertir que el grado de seguridad del WCMS auditado aún no se halla en su cota más óptima, calificándolo, en términos de seguridad web, como software de **nivel medio**.

5.2. Relación resultado-objetivo

Una forma eficaz y transparente de autoevaluar nuestro trabajo, consiste en contrastar los resultados obtenidos finalmente con los objetivos o propósitos marcados al comienzo de éste.

Empezaremos rememorando en primer lugar los **objetivos propuestos al comienzo** de este trabajo:

- Verificar la calidad de implementación del WCMS, analizándolo a nivel de código.
- Describir y justificar las fallas de seguridad detectadas sobre éste.
- Exponer las soluciones pertinentes, recomendadas por la OWASP, a fin de incrementar el nivel de seguridad del software que se audita.
- Crear conciencia y educar desarrolladores, diseñadores, arquitectos, gerentes y organizaciones, sobre las consecuencias de las vulnerabilidades web más destacadas.

Llegados a este punto, considero exitoso el cumplimiento de la gran mayoría de los objetivos previos marcados para esta auditoría.

Cabe mencionar que, si bien no depende exclusivamente de mí el crear conciencia y educación en materia de seguridad web, tal y como se expone en el último de los cuatro puntos anteriores, sí que espero haber contribuido con este trabajo, de forma positiva y transparente, a tal cometido.

En cuanto al análisis del código y su calidad de implementación, por razones expresas de los propietarios del software, no se muestran detalles textuales en la fase de actuación, lo que no exime la veracidad de los resultados obtenidos, en cuanto a detección de vulnerabilidades, ni la correcta interpretación y justificación de éstas, junto a las soluciones recomendadas que finalmente se han propuesto.

Con todo ello, se puede estimar perfectamente que la relación resultado-objetivo para este trabajo ha inferido de forma satisfactoria y eficaz.

5.3. Líneas futuras de investigación/desarrollo

Cabe admitir, como perspectiva de futuro, la posibilidad de aportar cierto valor añadido a este trabajo. Debemos recordar, como bien se mencionó en los capítulos previos, que no existe software plenamente seguro, que siempre surgirá una nueva puerta trasera o seguiremos siendo víctimas de nuestro despiste y dejadez como desarrolladores por mucho que nos empeñemos.

Como tal, hay que reconocer también, que al margen del Top 10 de OWASP estudiado para esta auditoría, existen innumerables vulnerabilidades por inspeccionar sobre nuestro WCMS. Vulnerabilidades que, en muchos casos a día de hoy, no se consideran oficialmente prioritarias o incluso se han extinguido ya con la evolución tecnológica, pero que en otros casos, sin embargo, se encuentran en fase de investigación (*alfa* o *beta*) y quizás con el tiempo se conviertan en vulnerabilidades potencialmente muy relevantes.

Por este motivo, debemos asumir que el abanico de pruebas de seguridad web sobre este software podría multiplicarse exponencialmente, pero que aun así no alcanzaríamos nunca el 100% de seguridad sobre el mismo.

Sí se podría tener en cuenta, también, un **mayor número de factores de investigación** sobre cada una de las 10 vulnerabilidades analizadas para esta auditoría. Con ello no solo conseguiríamos catalogar correctamente una u otra amenaza concreta, sino focalizar soluciones técnicas más objetivas que nos permitiesen erradicar dichas amenazas de una forma más ágil, transparente y personalizada.

Por último, sería interesante integrar una sección en la cual se hiciese especial hincapié, no solo en tratar de detectar y corregir posibles vulnerabilidades sobre este tipo de entornos, fin perseguido para esta auditoría, sino en prevenir el desarrollo deficiente de éstos desde etapas tempranas, empleando para ello un **modelo de ciclo de vida del software** acorde a los requerimientos específicos del producto que se desee implementar. De esta forma, conseguiríamos mitigar amplia y eficazmente el espectro de vulnerabilidades para futuras auditorías.

Referencias bibliográficas

[1] IMPORTANCIA DE LA SEGURIDAD INFORMATICA DENTRO DE LAS EMPRESAS. (2012, noviembre). *Revista RED (La comunidad de expertos en redes)*. Fecha de consulta: agosto 31, 2015 desde: http://seginfouno.blogspot.com.es/2011/06/seguridad-informatica-que-por-que-y.html

[2] Seguridad de aplicaciones web. (2015, 23 de abril). *Wikipedia, La enciclopedia libre.* Fecha de consulta: agosto 31, 2015 desde: https://es.wikipedia.org/w/index.php?title=Seguridad_de_aplicaciones_web&oldid=81989845

[3] Aspectos Básicos de la Seguridad en Aplicaciones Web. (2011, 26 de mayo). *Coordinación de Seguridad de la Información (CSI).* Fecha de consulta: agosto 31, 2015 desde: http://www.seguridad.unam.mx/documento/?id=17

[4] Vulnerabilidad. (2015, 04 de agosto). *Wikipedia, La enciclopedia libre.* Fecha de consulta: septiembre 01, 2015 desde: https://es.wikipedia.org/w/index.php?title=Vulnerabilidad&oldid=84222667

[5] Sistema de gestión de contenido web. (2015, 21 de abril). *Wikipedia, La enciclopedia libre.* Fecha de consulta: septiembre 01, 2015 desde: https://es.wikipedia.org/w/index.php?title=Sistema_de_gesti%C3%B3n_de_contenido_web&oldid=81950635

[6] Web vs. Gestor de contenidos. (2015, 11 de febrero). *Javier Arévalo*. Fecha de consulta: septiembre 02, 2015 desde: http://javierarevalo.com/blog/web-vs-gestor-de-contenidos/

[7] Seguridad. (2015). *webControl CMS*. Fecha de consulta: septiembre 02, 2015 desde: http://www.webcontrol.es/es/tecnologia/seguridad/#recurso

[8] Metodología. (2015, 09 de septiembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: septiembre 07, 2015 desde: https://es.wikipedia.org/w/index.php?title=Metodolog%C3%ADa&oldid=85017213

[9] Antonio Fuertes Maestro. Trabajo Fin de Máster (UNIR). (2014, 05 de marzo). *Elaboración de una metodología de test de intrusión dentro de la auditoría de seguridad*. Fecha de consulta: septiembre 07, 2015 desde: http://reunir.unir.net/handle/123456789/2331

[10] Open Web Application Security Project. (2013, 14 de marzo). *Wikipedia, La enciclopedia libre.* Fecha de consulta: septiembre 08, 2015 desde: https://es.wikipedia.org/w/index.php?title=Open_Web_Application_Security_Project&oldid=65191456

[11] Rubén Velasco. (2015, 25 de abril). *OWASP ZAP, herramienta para auditar la seguridad de una página web*. Fecha de consulta: septiembre 09, 2015 desde: http://www.redeszone.net/2015/04/25/seguridad-web-owasp-zap/

[12] Comprobación de la seguridad de los sistemas con un Test de intrusión. (2012). *A2SECURE*. Fecha de consulta: septiembre 09, 2015 desde: http://www.a2secure.com/auditorias/test-de-intrusion