
This is the **published version** of the bachelor thesis:

Shao, Bo. *Desarrollo de herramientas de auditoría técnica y hardening para servicios y aplicaciones web*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334414>

under the terms of the  license.

Desarrollo de herramientas de auditoría técnica y hardening para servicios y aplicaciones web

Bo Shao

Resumen– Este trabajo desarrolla una herramienta pensada para ayudar a desarrolladores web a detectar y corregir configuraciones deficientes de seguridad en servicios y aplicaciones accesibles por HTTP/HTTPS. La herramienta automatiza comprobaciones de hardening habituales, como el análisis de TLS/SSL y certificados, la revisión de cabeceras HTTP de seguridad y la detección de exposición de recursos o rutas sensibles. El objetivo es ofrecer resultados claros, con evidencias y recomendaciones priorizadas, para facilitar la corrección de problemas durante el desarrollo y el despliegue. La herramienta se ha probado mediante escenarios representativos para comprobar su utilidad y sus límites.

Palabras clave– Seguridad web; hardening; auditoría técnica; TLS/SSL; certificados digitales; cabeceras HTTP de seguridad; exposición de recursos sensibles.

Abstract– This work develops a tool designed to help web developers detect and fix weak security configurations in services and web applications accessible via HTTP/HTTPS. The tool automates common hardening checks, such as TLS/SSL and certificate analysis, reviewing security-related HTTP headers, and detecting exposure of sensitive resources or paths. The goal is to provide clear results, including evidence and prioritized recommendations, to facilitate issue remediation during development and deployment. The tool has been tested using representative scenarios to assess its usefulness and limitations.

Keywords– Web security; hardening; technical audit; TLS/SSL; digital certificates; security HTTP headers; exposure of sensitive resources.



1 INTRODUCCIÓN

EN la actualidad, las páginas y aplicaciones web siguen siendo uno de los vectores de ataque más importantes para los ciberdelincuentes [1]. Además, la seguridad de estas plataformas es una disciplina amplia y en constante evolución, ya que surgen nuevos ataques y vulnerabilidades [2]. Por este motivo, es necesario mantener de forma continua una configuración de seguridad adecuada.

En este contexto, la auditoría de seguridad de sitios web consiste en evaluar de manera sistemática los controles técnicos y la configuración de un servicio con el objetivo de prevenir, identificar y corregir posibles amenazas y debilidades de seguridad. Asimismo, estas auditorías contribuyen a verificar el cumplimiento de normativas y buenas prácticas del sector, y a reforzar la confianza y la

experiencia de los usuarios, al ayudar a mantener el sitio web seguro y fiable [1].

En escenarios reales, es común encontrar errores de configuración o ausencia de controles básicos, especialmente durante la puesta en producción o tras cambios en la infraestructura. Entre los problemas más habituales destacan el uso de configuraciones TLS no recomendadas y una gestión deficiente de certificados [4, 5], la falta de cabeceras HTTP de seguridad [6, 7] y la exposición accidental de ficheros o rutas sensibles, como ficheros de configuración o repositorios [8, 9]. Para mitigar estos riesgos, se suelen combinar medidas como la aplicación de guías de hardening y configuraciones base, la realización de auditorías periódicas y el uso de comprobaciones automatizadas para detectar desviaciones de forma repetible, especialmente tras cambios y despliegues [1, 3].

- E-mail de contacto: bo.shao@autonoma.cat
- Menció: Tecnologies de la Informació
- Trabajo tutorizado por: Joan Mata Pàrraga (DEIC)
- Curso 2025/26

2 ESTRUCTURA DEL INFORME

El documento se organiza de la siguiente manera:

En la **introducción** se presenta el contexto y la motivación del trabajo, justificando la necesidad de revisar de forma continua la seguridad de servicios y aplicaciones web accesibles por HTTP/HTTPS. Asimismo, sitúa el proyecto en el ámbito de la auditoría de configuración y el hardening.

En los **objetivos** se expone el propósito principal del trabajo y se describen los objetivos específicos abordados durante el diseño e implementación de la herramienta.

En el **estado del arte** se revisan conceptos y herramientas relacionadas con la auditoría técnica y el hardening web. En este apartado se presentan referencias como OWASP ZAP y Nikto, y se sitúa la herramienta desarrollada dentro de este contexto, explicando su enfoque y las diferencias principales respecto a soluciones más generales.

En la **metodología y planificación** se presenta el enfoque de trabajo seguido y la forma en que se han organizado las distintas tareas del proyecto. En este apartado se explica el desarrollo modular e incremental adoptado, el lenguaje y el entorno de desarrollo empleados, y la progresión seguida desde la definición inicial del trabajo hasta la implementación de la herramienta.

En el **desarrollo** se describe el trabajo realizado en la implementación de la herramienta. En particular, se presenta la estructura modular del proyecto, la construcción de la interfaz de línea de comandos, el motor de escaneo, los módulos de análisis desarrollados y la generación de informes en formatos estructurados.

En las **conclusiones** se resumen los resultados principales obtenidos, se valoran las limitaciones de la herramienta y se plantean posibles líneas de trabajo futuro.

En el apartado de **uso de la IA generativa** se explica cómo se ha utilizado esta tecnología como apoyo en la redacción, revisión e implementación del trabajo, dejando claro que el contenido final ha sido revisado y validado.

3 OBJETIVOS

En este apartado se presentan los objetivos del trabajo. Se expone el objetivo general de la herramienta y se desarrollan los objetivos específicos abordados durante su diseño e implementación, diferenciando entre la base funcional inicial y las ampliaciones incorporadas posteriormente.

3.1 OBJETIVO GENERAL

Este trabajo presenta el desarrollo de una herramienta CLI, es decir, una interfaz de línea de comandos, orientada a la auditoría técnica y al hardening de servicios y aplicaciones web accesibles por HTTP/HTTPS.

La primera fase se centró en construir la base funcional de la herramienta, incluyendo la arquitectura modular, la interfaz de línea de comandos, el motor de escaneo y los módulos TLS/SSL y cabeceras HTTP. A partir de esta base, la segunda fase amplía la herramienta con la detección de exposición de recursos sensibles y la generación de informes en formatos estructurados.

El objetivo general es mejorar la utilidad práctica de la herramienta, aumentando la cobertura de análisis y facilitando la interpretación, conservación y revisión de los resultados obtenidos durante las pruebas.

3.2 OBJETIVOS ESPECÍFICOS

Para ello, el trabajo se ha organizado en distintas fases, cada una orientada a ampliar progresivamente la funcionalidad de la herramienta. En primer lugar, se ha definido el alcance de la herramienta, limitando el análisis a servicios accesibles mediante HTTP/HTTPS a partir de un host, y orientando el proyecto a la auditoría de configuración y al hardening del servicio. De este modo, el trabajo se centra en comprobaciones no intrusivas sobre la configuración y la posible exposición de recursos, sin incluir técnicas de ataque ni explotación de vulnerabilidades.

En segundo lugar, se ha diseñado una arquitectura modular que permite separar la interfaz de ejecución, el motor de escaneo y los distintos módulos de comprobación. Este objetivo resulta importante para facilitar el desarrollo progresivo de la herramienta, mejorar la organización del código y permitir que nuevas funcionalidades puedan incorporarse más adelante sin modificar en exceso la estructura general del sistema.

Otro de los objetivos abordados en la primera fase ha sido la implementación de una base funcional de la herramienta, compuesta por una interfaz de línea de comandos y un motor de escaneo capaz de coordinar la ejecución de las comprobaciones disponibles y de presentar los resultados de forma estructurada. Con ello, se ha conseguido establecer el núcleo operativo necesario para integrar los módulos de análisis desarrollados.

En cuanto a las comprobaciones de seguridad iniciales, se ha trabajado en la implementación de un módulo centrado en TLS/SSL. El objetivo de este bloque ha sido permitir la revisión de aspectos básicos de la seguridad del canal de comunicación, incluyendo la conectividad con los puertos habituales, la validación del certificado, la revisión de su expiración y el análisis de las versiones TLS aceptadas por el servidor. Con esta parte se busca detectar configuraciones inseguras, certificados no válidos o próximos al vencimiento y otros problemas relacionados con la protección de las comunicaciones.

Asimismo, se ha desarrollado un módulo orientado a la revisión de cabeceras HTTP de seguridad. En este caso, el objetivo ha sido comprobar la presencia y configuración de mecanismos de protección relacionados con la política de respuesta del servidor y con la navegación segura, así

como analizar determinadas redirecciones que puedan resultar inseguras o poco recomendables. Esta parte permite identificar ausencias de cabeceras, configuraciones débiles o decisiones de hardening mejorables dentro de la configuración del servicio web.

En la segunda fase del proyecto se ha incorporado el módulo de exposición de recursos sensibles. Este objetivo amplía la cobertura de la herramienta más allá del análisis TLS/SSL y de cabeceras HTTP, permitiendo comprobar si existen ficheros, directorios o rutas sensibles accesibles públicamente. La exposición de este tipo de recursos puede revelar información interna de la aplicación, como ficheros de configuración, copias de seguridad, volcados de bases de datos, repositorios, documentación técnica o rutas de diagnóstico.

Por último, se ha incorporado la generación de informes en formatos JSON y HTML. Esta funcionalidad permite conservar los resultados de una ejecución, revisarlos posteriormente y presentar los hallazgos de forma más clara y estructurada. El formato JSON resulta útil para almacenar los resultados de manera procesable, mientras que el formato HTML facilita una revisión visual más cómoda de los hallazgos, sus evidencias y sus recomendaciones.

Como resultado, la herramienta integra tres dimensiones principales de hardening: la configuración TLS/SSL, las cabeceras HTTP de seguridad y la exposición accidental de recursos sensibles.

4 ESTADO DEL ARTE

La seguridad de aplicaciones y servicios web se apoya en buenas prácticas de hardening y en herramientas de análisis que permiten detectar configuraciones débiles. OWASP recoge la configuración incorrecta de seguridad como uno de los riesgos relevantes en aplicaciones web dentro de la categoría *Security Misconfiguration* [3]. Esta categoría se relaciona con el objetivo del presente trabajo, ya que la herramienta desarrollada se centra en la revisión de configuraciones de seguridad en servicios web.

Ya existen herramientas orientadas al análisis de seguridad web que sirven como referencia en este ámbito. OWASP ZAP permite realizar pruebas de seguridad sobre aplicaciones web con un alcance amplio, incluyendo análisis pasivos y activos [12]. Nikto, por su parte, se centra en la revisión de servidores web mediante la búsqueda de configuraciones inseguras, ficheros potencialmente peligrosos, componentes desactualizados y otros problemas habituales [13]. La existencia de estas herramientas muestra que la automatización de comprobaciones de seguridad es una práctica habitual, aunque su enfoque suele estar dirigido a auditorías más generales o avanzadas.

Frente a estos enfoques más amplios, el presente trabajo plantea una herramienta centrada en una revisión técnica básica de hardening web. La diferencia principal reside en ofrecer una CLI sencilla que permite ejecutar comprobaciones básicas de TLS/SSL, cabeceras HTTP y

recursos o rutas sensibles comunes, interpretar los resultados mediante evidencias y recomendaciones, y generar informes en JSON y HTML. Además, su diseño modular facilita la ampliación futura de la herramienta con nuevas comprobaciones.

5 METODOLOGÍA Y PLANIFICACIÓN

El desarrollo del proyecto se ha llevado a cabo siguiendo una metodología modular e incremental, orientada a implementar la herramienta por partes y a validar cada avance antes de integrarlo en el conjunto. La implementación se ha realizado en Python como lenguaje principal, utilizando Visual Studio Code como entorno de desarrollo, con el objetivo de mantener un código organizado, reutilizable y fácil de mantener.

Este enfoque ha permitido organizar el trabajo en varias fases. En una primera etapa se definió el alcance del proyecto y se preparó la memoria inicial, estableciendo los objetivos del trabajo y revisando información relacionada con auditoría técnica y hardening en servicios web. A continuación, se abordó el diseño general de la herramienta, definiendo una estructura que permitiese desarrollar y probar cada parte de forma separada antes de su integración.

Una vez establecida esta base, el trabajo se centró en la implementación progresiva de las distintas funcionalidades. Para ello, cada nueva comprobación se fue incorporando y ajustando gradualmente, realizando pruebas específicas sobre sitios web para comprobar tanto el comportamiento de las librerías utilizadas como la lógica implementada. Este proceso permitió detectar errores, ajustar el funcionamiento de las comprobaciones y revisar la claridad de las evidencias y recomendaciones generadas. Además, durante el desarrollo también se identificaron y documentaron las limitaciones observadas en cada caso.

Posteriormente, se amplió la herramienta sobre la base construida inicialmente. Las nuevas funcionalidades se integraron respetando la separación entre la interfaz de línea de comandos, el motor de escaneo, los módulos de análisis y la generación de informes. De este modo, se pudo ampliar el alcance de la herramienta sin alterar de forma significativa su estructura general.

6 DESARROLLO

En esta sección se describe la implementación realizada y se presentan los principales resultados obtenidos durante el desarrollo de la herramienta. En particular, se detallan la organización interna del proyecto, la construcción de la interfaz de línea de comandos, el motor de escaneo, el funcionamiento de los módulos de análisis desarrollados y la generación de informes en formatos estructurados.

En primer lugar, se ha definido una estructura organizada por paquetes para separar la interfaz de ejecución, el núcleo de auditoría y los distintos módulos de comproba-

ción. Esta organización permite mantener el código más claro, facilitar su ampliación y separar la lógica de análisis de la forma en que el usuario ejecuta la herramienta. En concreto, se ha diferenciado la capa CLI, encargada del punto de entrada del programa, de la capa CORE, donde se concentra la coordinación del escaneo y la implementación de las comprobaciones (Fig. 1).

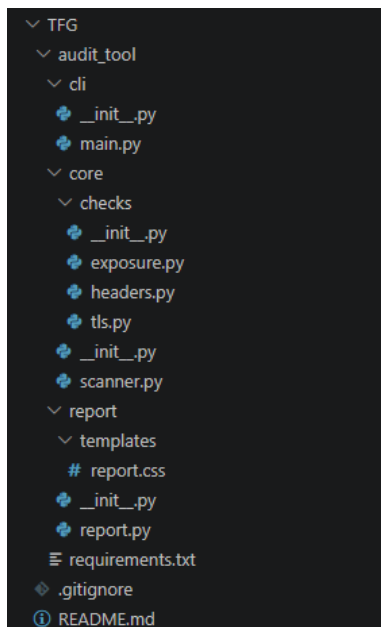


Fig. 1: Arquitectura

A partir de esta estructura, se ha implementado una CLI funcional. Esta interfaz permite al usuario indicar el objetivo a analizar, seleccionar el módulo concreto que desea ejecutar (*headers*, *tls*, *exposure* o *all*) y configurar el tiempo máximo de espera de las peticiones. Para esta parte se ha utilizado principalmente la librería estándar “*argparse*”, que permite definir argumentos posicionales y opcionales y construir una interfaz de línea de comandos sencilla y mantenible [10].

Sobre esta base se ha implementado también el motor de escaneo. Este componente coordina la ejecución de la herramienta: recibe el objetivo introducido por el usuario, determina qué módulo debe ejecutarse y reúne los resultados obtenidos. Para que todos los módulos puedan tratarse de la misma forma, los resultados se devuelven mediante una estructura común basada en hallazgos (*findings*). Cada hallazgo incluye un título, una severidad, una descripción, una evidencia y una recomendación. Esta estructura facilita la presentación de resultados por consola y permite reutilizar la misma información para generar informes en JSON y HTML.

El siguiente fragmento muestra la plantilla común utilizada por la herramienta para representar un hallazgo:

```
finding = {
    "title": "",
    "severity": "",
    "description": "",
    "evidence": "",
    "recommendation": ""
}
```

6.1 MÓDULO TLS/SSL

En cuanto a los módulos de auditoría, se ha implementado el módulo TLS/SSL. Este módulo revisa aspectos básicos de la conexión segura entre el cliente y el servidor. Para ello, comprueba si los puertos web habituales están accesibles, establece una conexión TLS con el servicio HTTPS y obtiene información del certificado presentado por el servidor. Estas comprobaciones son importantes porque una configuración TLS incorrecta puede afectar a la confidencialidad de los datos, a la integridad de la comunicación y a la confianza en la identidad del sitio web [4, 5].

La primera comprobación consiste en verificar si el servicio responde por HTTP, por HTTPS o por ambos. Aunque es una comprobación sencilla, sirve para entender el contexto del análisis y decidir qué verificaciones posteriores pueden realizarse.

Entre las comprobaciones desarrolladas se incluyen la validación básica del certificado, la obtención de información sobre el sujeto y el emisor, la revisión de la fecha de expiración y la detección de proximidad al vencimiento. Estas verificaciones permiten identificar certificados no válidos, caducados o próximos a caducar, situaciones que pueden provocar errores de confianza en el cliente, interrupciones del servicio o dificultades para garantizar que el servidor es realmente quien dice ser. Del mismo modo, la obtención de información sobre el certificado permite contextualizar mejor el resultado del análisis y verificar si la identidad presentada por el servidor es coherente con el objetivo examinado.

Asimismo, se analiza la versión TLS negociada y el cifrado utilizado, y se realizan pruebas para determinar si el servidor acepta versiones antiguas o no recomendadas. Esta comprobación permite evaluar el nivel real de protección ofrecido por el servicio, ya que no basta con que exista soporte para TLS, sino que es necesario verificar qué versión del protocolo y qué parámetros criptográficos se utilizan finalmente durante la negociación. El uso de protocolos obsoletos o configuraciones criptográficas débiles puede reducir la robustez de la conexión y facilitar ataques sobre el canal de comunicación.

En el caso del módulo TLS/SSL, como se muestra en la Fig. 2, la herramienta analiza un objetivo HTTPS mediante el comando `python -m audit_tool.cli.main https://example.com --module tls`. Esta prueba comprueba la conectividad segura, el estado del certificado y las versiones TLS aceptadas por el servidor. El resultado muestra una conexión TLS válida, un certificado vigente y soporte para versiones actuales del protocolo. La evidencia incluye la información del certificado, la versión TLS negociada y el cifrado utilizado. Al tratarse de una configuración correcta, los hallazgos son principalmente informativos, lo que demuestra que la herramienta también distingue escenarios adecuados y no solo situaciones problemáticas.

```

PS C:\Users\BShao\Documents\INFO\INFO> python -m audit_tool.cli.main https://example.com --module tls
--- RESULTADO DEL ESCANEO ---
Objetivo: https://example.com
Módulo: tls
Hallazgos: 8
Errores: 0
--- Hallazgos ---

[1] Servicios web HTTP y HTTPS disponibles
Severidad: Info
Descripción: El objetivo expone servicios web en los puertos 80 y 443
Evidencia: example.com:80 accesible | example.com:443 accesible
Recomendación: Comprobar si el tráfico HTTP redirige correctamente a HTTPS

[2] Conexión TLS establecida
Severidad: Info
Descripción: Se ha podido establecer una conexión TLS con el objetivo en el puerto 443
Evidencia: Puerto: 443 | Versión TLS negociada: TLSv1.3 | Cipher: TLS_AES_256_GCM_SHA384
Recomendación: Verificar que la versión y los cifrados negociados cumplen las políticas de seguridad

[3] Cipher negociado sin indicadores débiles evidentes
Severidad: Info
Descripción: No se detectaron patrones típicos de debilidad en el cipher negociado
Evidencia: Cipher: TLS_AES_256_GCM_SHA384
Recomendación: Revisar también el conjunto completo de suites soportadas si se desea un análisis más exhaustivo

[4] Información del certificado
Severidad: Info
Descripción: Se obtuvo información básica del certificado presentado por el servidor en el puerto 443
Evidencia: Subject: commonName=example.com
Recomendación: Comprobar que el certificado corresponde al dominio esperado

[5] Información del emisor del certificado
Severidad: Info
Descripción: Se obtuvo información básica del emisor del certificado
Evidencia: Issuer: countryName=US, organizationName=CLOUDFLARE, INC., commonName=Cloudflare TLS Issuing ECC CA 1
Recomendación: Comprobar que el certificado fue emitido por una entidad de confianza

[6] Certificado vigente
Severidad: Info
Descripción: El certificado TLS del servidor en el puerto 443 no está expirado
Evidencia: Fecha de expiración: 2026-07-01T21:24:46+00:00 | Días restantes: 80
Recomendación: Supervisar la fecha de expiración y renovar antes de su vencimiento

[7] Soporte de TLS 1.2
Severidad: Info
Descripción: El servidor acepta TLS 1.2
Evidencia: example.com:443 acepta TLS 1.2
Recomendación: Mantener únicamente versiones modernas si es posible

[8] Soporte de TLS 1.3
Severidad: Info
Descripción: El servidor acepta TLS 1.3
Evidencia: example.com:443 acepta TLS 1.3
Recomendación: Mantener únicamente versiones modernas si es posible

```

Fig. 2: Salida del módulo TLS/SSL

Desde el punto de vista del hardening, este módulo permite identificar problemas que pueden afectar directamente a la confianza del canal seguro, como certificados caducados, próximos a vencer o configuraciones TLS obsoletas. Estos hallazgos no solo indican posibles debilidades técnicas, sino también aspectos operativos que deben mantenerse bajo control para evitar interrupciones del servicio o advertencias de seguridad en el navegador.

Durante las pruebas del módulo TLS/SSL se detectó una limitación en la recuperación de información del certificado cuando la validación fallaba. Para resolver este caso, se incorporó un mecanismo adicional que permite obtener y decodificar temporalmente el certificado en formato PEM, lo que posibilita seguir analizando datos relevantes incluso cuando la validación no se completa correctamente.

Para desarrollar este módulo se han utilizado varias librerías de la biblioteca estándar de Python. En particular, “*socket*” se ha empleado para comprobar la conectividad con los puertos del servicio y establecer conexiones TCP; “*ssl*” se ha utilizado para realizar la negociación TLS, obtener información del certificado y probar versiones concretas del protocolo; “*datetime*” se ha usado para procesar la fecha de expiración del certificado y calcular los días restantes hasta su vencimiento; “*urllib.parse*” se ha utilizado para normalizar el objetivo y extraer correctamente el host; y “*tempfile*” junto con “*os*” se han empleado para almacenar temporalmente certificados en formato PEM y poder decodificarlos cuando la validación falla pero aun así se desea inspeccionar su contenido [10].

6.2 MÓDULO DE CABECERAS HTTP

También se ha implementado el módulo de cabeceras HTTP de seguridad. Las cabeceras HTTP son instrucciones

que el servidor envía al navegador junto con la respuesta de una página. Algunas de ellas permiten reforzar la seguridad del sitio, por ejemplo obligando a usar HTTPS, limitando la carga de contenido externo o evitando que la página se incruste dentro de otros sitios. Por este motivo, el módulo revisa redirecciones de HTTP a HTTPS, posibles redirecciones hacia dominios externos y cabeceras como *Strict-Transport-Security*, *Content-Security-Policy*, *X-Frame-Options*, *X-Content-Type-Options*, *Referrer-Policy* y *Permissions-Policy* [6, 7].

Por ejemplo, la cabecera *Strict-Transport-Security* permite indicar al navegador que debe acceder siempre al sitio mediante HTTPS, evitando que futuras conexiones se realicen por HTTP sin cifrar. Si esta protección no está presente, pueden producirse accesos iniciales sin cifrado o usos de conexiones menos seguras. Del mismo modo, *X-Frame-Options* protege frente a ataques de *clickjacking*, impidiendo que la aplicación pueda cargarse dentro de un *frame* o *iframe* malicioso para engañar al usuario y forzar interacciones no deseadas.

Asimismo, *X-Content-Type-Options* con el valor *nosniff* evita que el navegador intente interpretar un recurso como un tipo de contenido distinto al declarado por el servidor. Si esta cabecera no se configura correctamente, el navegador podría procesar ciertos recursos de forma insegura. En el caso de *Content-Security-Policy*, esta cabecera permite restringir de qué orígenes pueden cargarse scripts, estilos, imágenes u otros recursos, reduciendo así el riesgo de que se ejecute contenido no autorizado dentro de la aplicación. Por otro lado, *Referrer-Policy* y *Permissions-Policy* permiten limitar, respectivamente, la información de navegación que se comparte entre peticiones y el acceso del navegador a determinadas funcionalidades sensibles.

En conjunto, la ausencia o mala configuración de estas cabeceras puede facilitar ataques como el *clickjacking*, la carga de contenido no controlado, la interpretación insegura de determinados recursos o la exposición innecesaria de información de navegación. Aun así, estos resultados deben interpretarse dentro del contexto de cada aplicación. No todas las cabeceras tienen la misma importancia en todos los casos, ya que su impacto depende del tipo de servicio, de los recursos que carga la página y de las funcionalidades expuestas al usuario. Por ello, el módulo no debe entenderse como una lista absoluta de requisitos, sino como una ayuda para detectar configuraciones ausentes, débiles o revisables dentro del proceso de hardening.

Para el módulo de cabeceras HTTP, como se observa en la Fig. 3, la herramienta analiza un objetivo HTTP mediante el comando `python -m audit_tool.cli.main http://sitio.com --module headers`. Esta prueba comprueba la presencia de cabeceras de seguridad y el comportamiento de las redirecciones. El resultado permite identificar qué mecanismos de protección están presentes y cuáles aparecen ausentes o son mejorables. En la salida se identifica la presencia de varias cabeceras de seguridad, aunque también se detecta la ausencia de *Permissions-Policy* y una redirección hacia un dominio externo. La evidencia se basa en las cabeceras devueltas

La exposición de recursos sensibles puede tener un impacto relevante en la seguridad de una aplicación web. Por ejemplo, un repositorio `.git` accesible públicamente puede revelar parte del código fuente o información del historial del proyecto. Un fichero `.env` puede contener credenciales, claves de API o datos de conexión a bases de datos. Del mismo modo, una copia de seguridad, un volcado SQL o una clave privada expuesta podrían comprometer información interna o facilitar accesos no autorizados.

Entre las rutas comprobadas se incluyen ficheros como `/.git/HEAD`, `/.git/config`, `/.env`, `/web.config`, `/.ssh/id_rsa`, `/.htpasswd`, `/backup.zip`, `/backup.tar.gz` y `/database.sql`. Estos recursos pueden revelar información interna si se encuentran accesibles públicamente.

Además, se comprueban directorios habituales como `/.git/`, `/backup/`, `/backups/`, `/uploads/`, `/files/`, `/logs/` o `/tmp/`, con el objetivo de detectar posibles listados de directorio habilitados. Por último, también se analiza el fichero `/robots.txt`, ya que puede contener referencias a zonas de administración, copias de seguridad, configuraciones o bases de datos.

El módulo no considera suficiente que una ruta devuelva un código de estado 200 OK. Para evitar falsos positivos, cada recurso tiene asociados indicadores específicos. En ficheros de texto se buscan cadenas características, como `ref: refs/heads/` para repositorios Git o términos como `db_password`, `secret_key` o `database_url` para ficheros de configuración. En el caso de archivos comprimidos, también se comprueban firmas binarias, como las cabeceras propias de ficheros ZIP o GZIP.

Una decisión técnica importante ha sido no limitar el análisis únicamente al código de estado HTTP devuelto por el servidor. Para reducir falsos positivos, el módulo incorpora indicadores específicos de contenido, comparación con respuestas base y firmas de respuestas normalizadas. Además, la herramienta realiza las solicitudes en modo *stream* y analiza solo una muestra inicial del cuerpo de la respuesta, evitando descargar por completo archivos potencialmente grandes, como copias de seguridad o volcados de bases de datos. Con ello, el análisis se mantiene dentro de un enfoque prudente y no intrusivo, a la vez que mejora la fiabilidad de los hallazgos generados.

Los hallazgos generados por este módulo se clasifican según su impacto potencial. La exposición de credenciales, repositorios Git, claves privadas, ficheros de autenticación, copias de seguridad o volcados SQL se considera de severidad alta. Los listados de directorio se clasifican normalmente como severidad media, ya que pueden revelar archivos internos o recursos no enlazados directamente desde la aplicación. Por otro lado, las referencias potencialmente sensibles en `robots.txt` se clasifican como severidad baja, ya que no implican una exposición directa, pero pueden facilitar tareas de reconocimiento.

Desde el punto de vista del hardening, este módulo

Fig. 3: Salida del módulo de cabeceras HTTP de seguridad

Durante las pruebas de este módulo se observó que algunas respuestas no reflejaban directamente el contenido real de la aplicación, sino páginas de error, bloqueo o protección intermedia. Para identificar esta situación, el módulo tuvo en cuenta aquellos casos en los que la respuesta HTTP devolvía un código 202 o un código de estado mayor o igual que 400. Esta condición resulta relevante porque las cabeceras presentes en ese tipo de respuestas pueden no corresponder a la configuración final del servicio analizado. Por ello, se incorporó esta comprobación en la lógica del módulo con el fin de contextualizar determinados hallazgos y evitar interpretaciones erróneas.

En este módulo se ha empleado la librería externa “*requests*”, que facilita el envío de peticiones HTTP/HTTPS, la gestión de redirecciones y el acceso a las cabeceras devueltas por el servidor. Además, se ha utilizado “*urllib.parse*” para construir y normalizar URLs, así como para comparar dominios originales y dominios finales en la cadena de redirecciones [10, 11].

Otro componente desarrollado es el módulo de exposición de recursos sensibles. Este módulo comprueba si una aplicación web deja accesibles desde Internet ficheros, directorios o rutas que deberían permanecer privados. La comprobación se realiza de forma controlada, mediante peticiones HTTP a una lista limitada de rutas habituales asociadas a errores de configuración o despliegue. De este modo, el análisis mantiene un enfoque prudente y no intrusivo.

aporta valor porque permite detectar errores de despliegue, descuidos en la organización de archivos o recursos ubicados en rutas que no deberían ser públicas. Un servicio puede funcionar correctamente para el usuario final y, al mismo tiempo, exponer ficheros internos o rutas residuales generadas durante el desarrollo. La detección temprana de estos recursos ayuda a corregir configuraciones inseguras antes de que puedan ser aprovechadas para obtener información sensible, preparar ataques posteriores o comprometer credenciales.

Para desarrollar este módulo se ha empleado la librería externa “*requests*”, utilizada para realizar peticiones HTTP/HTTPS a las rutas candidatas y leer las respuestas en modo *stream*. Además, se han utilizado funciones de la biblioteca estándar “*urllib.parse*” para normalizar el objetivo, construir las bases de análisis y generar correctamente las URLs de las rutas candidatas. También se ha utilizado “*hashlib*” para calcular firmas SHA-256 de respuestas normalizadas y “*secrets*” para generar rutas aleatorias inexistentes que permiten construir una respuesta base y reducir falsos positivos [10, 11].

```
PS C:\Users\BShao\Documents\TFG\TFD> python -m audit_tool.cli.main http://localhost:8080 --module exposure

===== RESULTADO DEL ESCANEO =====
Objetivo: http://localhost:8080
Módulo: exposure
Hallazgos: 4
Errores: 0
---- Hallazgos ----

[1] Repositorio Git expuesto
Severidad: high
Descripción: El fichero .git/HEAD es accesible desde HTTP/HTTPS, lo que indica una posible exposición del repositorio Git
Evidencia: URL: http://localhost:8080/.git/HEAD | Status code: 200 | Content-Type: application/octet-stream | Content-Length: 20 | Indicadores detectados: refs/heads/
Recomendación: Bloquear el acceso al directorio .git desde el servidor web y revisar si el repositorio completo ha quedado expuesto

[2] Fichero .env potencialmente expuesto
Severidad: high
Descripción: El fichero .env parece accesible públicamente y podría contener credenciales o secretos de configuración
Evidencia: URL: http://localhost:8080/.env | Status code: 200 | Content-Type: application/octet-stream | Content-Length: 150 | Indicadores detectados: db_password, database_url, secret_key
Recomendación: Mover el fichero fuera del directorio público, revocar secretos expuestos y bloquear su acceso desde el servidor web

[3] Dump SQL expuesto
Severidad: high
Descripción: Se ha detectado un posible volcado de base de datos accesible públicamente
Evidencia: URL: http://localhost:8080/database.sql | Status code: 200 | Content-Type: application/sql | Content-Length: 150 | Indicadores detectados: create table, insert into
Recomendación: Eliminar dump SQL del directorio público y rotar credenciales si contienen datos sensibles

[4] robots.txt revela rutas potencialmente sensibles
Severidad: low
Descripción: El fichero robots.txt contiene referencias a rutas que podrían revelar zonas internas o recursos sensibles
Evidencia: URL: http://localhost:8080/robots.txt | Status code: 200 | Content-Type: text/plain; charset=utf-8 | Content-Length: 92 | Indicadores detectados: .env, admin, backup, database.sql
Recomendación: Evitar usar robots.txt como mecanismo de ocultación de rutas sensibles y proteger dichas rutas con controles reales de acceso
```

Fig. 4: Salida del módulo de exposición de recursos

En la prueba del módulo de exposición de recursos sensibles, como se muestra en la Fig. 4, la herramienta analiza un entorno local controlado mediante el comando `python -m audit_tool.cli.main http://localhost:8080 --module exposure`. En este escenario se expusieron deliberadamente recursos ficticios para comprobar la capacidad de detección del módulo. El resultado muestra que la herramienta identifica rutas sensibles como `/.git/HEAD`, `/.env`, `/database.sql` o `/robots.txt`. La evidencia incluye la URL detectada y los indicadores de contenido que justifican el hallazgo. La severidad es alta en los recursos que pueden revelar código fuente, credenciales o información de base de datos, mientras que las referencias potencialmente sensibles en `robots.txt` se consideran de severidad baja. Esta prueba demuestra que el módulo puede detectar exposiciones introducidas de forma controlada y clasificarlas según su impacto potencial.

6.4 GENERACIÓN DE INFORMES

Finalmente, se ha incorporado la generación de informes en formatos estructurados. Esta funcionalidad permite guardar

los resultados de una ejecución y revisarlos posteriormente sin depender solo de la salida por consola. De este modo, la herramienta resulta más útil para documentar pruebas, comparar ejecuciones y conservar evidencias de los hallazgos detectados.

Los informes reutilizan la misma estructura común de hallazgos empleada por los módulos de análisis. Esto permite presentar de forma homogénea los resultados procedentes del análisis TLS/SSL, de las cabeceras HTTP y de la exposición de recursos sensibles.

Se han implementado dos formatos principales de salida: JSON y HTML. El informe JSON conserva la estructura interna de los resultados, incluyendo el objetivo analizado, el módulo ejecutado, los hallazgos obtenidos y los posibles errores. Este formato resulta útil para almacenar los datos de forma procesable y facilitar su reutilización posterior.

Por otro lado, el informe HTML está orientado a la lectura humana. Presenta la información de forma más clara y visual, incluyendo datos generales del escaneo, un resumen de riesgos por severidad y el detalle de cada hallazgo con su descripción, evidencia y recomendación. Además, el diseño del informe se ha preparado para que pueda abrirse desde un navegador y exportarse manualmente a PDF, facilitando su entrega o conservación como documento final.

Para desarrollar esta funcionalidad se han utilizado varias librerías de la biblioteca estándar de Python. En particular, `json` se ha empleado para serializar los resultados en formato JSON; `datetime` se ha utilizado para incluir la fecha de generación del informe; `html.escape` se ha usado para escapar el contenido insertado en el HTML; y `pathlib` se ha utilizado para gestionar rutas y la carga del archivo de estilos CSS separado del código principal del generador de informes. Además, se ha ampliado el uso de `argparse` en el punto de entrada de la CLI para incorporar los argumentos opcionales `--report` y `--output`, permitiendo seleccionar el formato del informe y definir la ruta de salida donde se guardará el archivo generado [10].

Audit Tool

Informe de auditoría técnica web

URL	http://localhost:8080	Módulo	all
Dominio	localhost	Hallazgos	13 totales
Fecha	2026-05-23 21:40:12	Errores	0

Resumen de Riesgos

3 ALTO	4 MEDIO	3 BAJO	3 INFO
--------	---------	--------	--------

Hallazgos Detectados

INFO 1. Respuesta HTTP recibida

Descripción: Se obtuvo una respuesta del objetivo para analizar sus cabeceras

Evidencia: URL: http://localhost:8080/ | Status code: 200

Recomendación: Usar esta evidencia como contexto para interpretar el resto de hallazgos

MEDIO 2. Sin redirección de HTTP a HTTPS

Descripción: No se observó una redirección automática de HTTP a HTTPS

Evidencia: URL inicial: http://localhost:8080 | Status code: 200

Recomendación: Redirigir automáticamente todo el tráfico HTTP hacia HTTPS

INFO 3. HSTS no evaluable sobre HTTP

Descripción: La cabecera Strict-Transport-Security solo aplica a respuestas servidas por HTTPS

Evidencia: No se observaron respuestas HTTPS en la cadena analizada

Recomendación: Comprobar HSTS sobre la versión HTTPS del servicio

Fig. 5: Informe HTML generado por la herramienta

Por último, como se ve en la Fig. 5, la herramienta genera un informe HTML tras ejecutar todos los módulos sobre un entorno local controlado mediante el comando `python -m audit_tool.cli.main http://localhost:8080 --module all --report html`. El informe presenta los datos generales del análisis, el dominio evaluado, el módulo ejecutado, el número total de hallazgos, los errores detectados y el resumen de riesgos por severidad. Además, incluye el detalle de cada hallazgo con su descripción, evidencia y recomendación. Esta prueba demuestra que los resultados pueden conservarse en un formato estructurado y revisarse posteriormente fuera de la consola.

7 CONCLUSIONES

El trabajo realizado ha permitido desarrollar una herramienta CLI orientada a la auditoría técnica y al hardening de servicios y aplicaciones web accesibles mediante HTTP/HTTPS. A partir de una arquitectura modular, se ha construido una base funcional capaz de coordinar distintos módulos de análisis y presentar los resultados mediante una estructura común de hallazgos. Esta organización ha facilitado la incorporación progresiva de nuevas funcionalidades y ha permitido mantener separadas la interfaz de ejecución, el motor de escaneo, los módulos de comprobación y la generación de informes.

Durante el desarrollo se han implementado tres bloques principales de análisis: el módulo TLS/SSL, el módulo de cabeceras HTTP de seguridad y el módulo de exposición de recursos sensibles. En conjunto, estos módulos permiten revisar aspectos relevantes de la configuración de seguridad de un servicio web, como la protección del canal de comunicación, la presencia de mecanismos de defensa a nivel HTTP y la posible exposición accidental de ficheros o rutas internas. De este modo, la herramienta cubre varias dimensiones habituales del hardening web desde un enfoque no intrusivo.

También se ha incorporado la generación de informes en formatos JSON y HTML. Esta funcionalidad mejora la utilidad práctica de la herramienta, ya que permite conservar los resultados de una ejecución, revisarlos posteriormente y presentar los hallazgos de forma clara. Además, la agrupación por severidad ayuda a priorizar las acciones correctivas, diferenciando entre resultados informativos, configuraciones mejorables y problemas con mayor impacto potencial.

No obstante, los resultados obtenidos deben interpretarse siempre dentro del contexto de cada aplicación. La ausencia de una cabecera, una redirección concreta o un determinado mecanismo de protección no implica necesariamente la existencia de una vulnerabilidad explotable de forma directa, sino que puede señalar una configuración revisable o una desviación respecto a buenas prácticas de hardening. Del mismo modo, no todas las aplicaciones requieren exactamente las mismas medidas, ya que estas pueden depender de la arquitectura, del entorno de despliegue o de decisiones técnicas justificadas.

Como trabajo futuro, sería conveniente ampliar las comprobaciones implementadas, incorporando nuevas revisiones de hardening y aumentando la cobertura de análisis de la herramienta. También sería útil validar su funcionamiento sobre un conjunto más amplio de objetivos representativos, mejorar la presentación final de los informes y organizar de forma más completa los registros y las evidencias generadas.

USO DE LA IA GENERATIVA

Durante el desarrollo de este trabajo se ha utilizado IA generativa como herramienta de apoyo en tareas concretas de redacción, revisión y mejora del documento, así como en consultas puntuales relacionadas con la implementación. Su uso se ha centrado principalmente en mejorar la claridad de algunos textos, reorganizar explicaciones, revisar el tono académico de la memoria y obtener orientación sobre posibles formas de expresar determinados aspectos técnicos.

En relación con la parte de implementación, la IA se ha empleado como apoyo para contrastar ideas, revisar fragmentos de código, detectar posibles errores y proponer alternativas de organización o mejora. No obstante, el código incorporado al proyecto ha sido desarrollado, revisado, adaptado y probado manualmente antes de formar parte de la herramienta. Las decisiones técnicas finales, la selección de funcionalidades, la estructura del proyecto y la interpretación de los resultados han sido realizadas por el autor del trabajo.

Asimismo, la IA generativa no se ha utilizado para inventar resultados, evidencias, pruebas ni conclusiones. Los resultados presentados en la memoria proceden de ejecuciones reales de la herramienta y de capturas obtenidas durante el proceso de validación. Las fuentes bibliográficas utilizadas también han sido revisadas y seleccionadas de forma manual, evitando incluir referencias no contrastadas.

Por tanto, el uso de IA en este proyecto debe entenderse como un recurso complementario de apoyo a la redacción, revisión y desarrollo, pero no como un sustituto del trabajo realizado. Todo el contenido final ha sido supervisado y ajustado de acuerdo con los objetivos del TFG, el funcionamiento real de la herramienta y las evidencias obtenidas durante las pruebas.

REFERENCIAS

- [1] SentinelOne, “Auditoría de seguridad de aplicaciones web: identificar y corregir vulnerabilidades,” SentinelOne Cybersecurity 101, updated Jul. 24, 2025. [Online]. Disponible: <https://www.sentinelone.com/es/cybersecurity-101/ciberseguridad/>

- [2] Cloudflare, “Inicio centro de aprendizaje,” Cloudflare Learning Center. [Online]. Disponible: <https://www.cloudflare.com/es-es/learning/>
- [3] OWASP, “A05:2021 – Security Misconfiguration,” OWASP. [Online]. Disponible: https://owasp.org/Top10/2021/A05_2021-Security_Misconfiguration/
- [4] S. Jain, “Su guía sobre ataques a certificados SSL y TLS,” Encryption Consulting, Feb. 23, 2025. [Online]. Disponible: <https://www.encryptionconsulting.com/es/Ataques-a-certificados-SSL-y-TLS/>
- [5] eMudhra Limited, “SSL/TLS Demystified: Your Ultimate Guide to Web Security,” eMudhra Blog, Jan. 3, 2024. [Online]. Disponible: <https://emudhra.com/en/blog/ssl-tls-demystified-your-ultimate-guide-to-web-security>
- [6] Invicti, “HTTP Security Headers: An Easy Way To Harden Your Web Applications,” Invicti Blog, n.d. [Online]. Disponible: <https://www.invicti.com/blog/web-security/http-security-headers>
- [7] S. Singh and I. Birlica, “Essential HTTP Headers for securing your web server,” Pentest-Tools.com Blog, Jul. 7, 2023 (updated Aug. 14, 2025). [Online]. Disponible: <https://pentest-tools.com/blog/essential-http-security-headers>
- [8] LabEx, “Localizar archivos web.config sensibles: Ciberseguridad,” LabEx Tutorials, n.d. [Online]. Disponible: <https://labex.io/es/tutorials/wireshark-how-to-locate-sensitive-web-config-files-419461>
- [9] Osint.com.ar, “Riesgos de un .git expuesto,” Osint.com.ar, n.d. [Online]. Disponible: <https://osint.com.ar/riesgos-de-un-git-expuesto/>
- [10] Python Software Foundation, “The Python Standard Library,” Python Documentation. [Online]. Disponible: <https://docs.python.org/es/dev/library/>
- [11] Python Package Index, “requests,” PyPI. [Online]. Disponible: <https://pypi.org/project/requests/>
- [12] OWASP ZAP, “ZAP Documentation,” Zed Attack Proxy. [Online]. Disponible: <https://www.zaproxy.org/>
- [13] CIRT.net, “Nikto,” Nikto Documentation. [Online]. Disponible: <https://cirt.net/nikto/>

APÉNDICE

RESUMEN DE USO

La herramienta se ejecuta desde la línea de comandos indicando el objetivo que se desea analizar y el módulo correspondiente.

La estructura general de ejecución es:

```
python -m audit_tool.cli.main
<objetivo>--module <módulo>
```

Los módulos disponibles son:

tls: analiza la configuración TLS/SSL y el certificado del servidor.

headers: revisa cabeceras HTTP de seguridad y redirecciones.

exposure: comprueba la posible exposición de rutas o recursos sensibles.

all: ejecuta todos los módulos disponibles.

Ejecutar el módulo TLS/SSL:

```
python -m audit_tool.cli.main
https://example.com --module tls
```

Ejecutar el módulo de cabeceras HTTP:

```
python -m audit_tool.cli.main
http://sitio.com --module headers
```

Ejecutar el módulo de exposición de recursos:

```
python -m audit_tool.cli.main
http://localhost:8080 --module exposure
```

Ejecutar todos los módulos:

```
python -m audit_tool.cli.main
http://localhost:8080 --module all
```

Además del módulo, pueden utilizarse argumentos opcionales:

--report: genera un informe en formato json o html.

--output: indica la ruta de salida donde se guardará el informe generado.

--timeout: define el tiempo máximo de espera de las peticiones en segundos.

Generar un informe HTML:

```
python -m audit_tool.cli.main
http://localhost:8080 --module all
--report html
```

Generar un informe JSON:

```
python -m audit_tool.cli.main
http://localhost:8080 --module all
--report json
```

Guardar el informe en una ruta de salida:

```
python -m audit_tool.cli.main
http://localhost:8080 --module all
--report html --output reports
```

La opción --output permite indicar una ruta de salida relativa o absoluta. Si no se indica, el informe se

guarda en la ubicación actual con un nombre generado automáticamente. También pueden utilizarse rutas como `../reports`.

Configurar el tiempo máximo de espera:

```
python -m audit_tool.cli.main  
http://localhost:8080 --module all  
--timeout 10
```

El resultado puede consultarse por consola o guardarse como informe JSON o HTML.