Informe de Auditoría de Ciberseguridad – WebGoat

Contents

[Ámbito y Alcance de la Auditoría 2](#_Toc153129700)

[Informe Ejecutivo 3](#_Toc153129701)

[a. Breve resumen del proceso realizado 3](#_Toc153129702)

[b. Vulnerabilidades destacadas 3](#_Toc153129703)

[c. Conclusiones 3](#_Toc153129704)

[d. Recomendaciones 3](#_Toc153129705)

[Descripción del Proceso de Auditoría 4](#_Toc153129706)

[a. Reconocimiento/Information gathering 4](#_Toc153129707)

[b. Explotación de vulnerabilidades detectadas 7](#_Toc153129708)

[c. Post-explotación 23](#_Toc153129709)

[d. Posibles mitigaciones 24](#_Toc153129710)

[e. Herramientas utilizadas 24](#_Toc153129711)

# Ámbito y Alcance de la Auditoría

El presente documento describirá las prácticas llevadas a cabo en el entorno de aprendizaje WebGoat utilizando la versión de Linux Kali con el objetivo de practicar y aprender los conocimientos en seguridad web necesarios para entender las herramientas y procesos básicos de ciberseguridad. Las prácticas incluyen una serie de vulnerabilidades asociados a lo que se denomina el OWASP Top 10, es decir las vulnerabilidades más comunes en el mundo de la ciberseguridad. Siguiendo los ejercicios, se has explorado estas vulnerabilidades comunes en aplicaciones web, como inyecciones SQL, ataques XSS (Cross-Site Scripting), y ataques de XML External Entity (XXE), entre otros. Igualmente se practica un último ejercicio para entender la importancia que tiene crear una contraseña que se considere “fuerte”, utilizando una combinación de caracteres y entendiendo lo rápido que se puede utilizar un programa con diccionario para averiguar una contraseña débil.

El objetivo has sido aprender, practicar y entender cuales son estas vulnerabilidades y porqué son tan comunes y utilizadas. Sabiendo el porqué conlleva también entender las formas en las que podemos protegernos de posibles ataques que utilicen estas vías y las consecuencias de lo que puede ocurrir si no buscamos formas de evitarlos.

En este informe, se presentarán los resultados obtenidos en cada una de las prácticas mencionadas, destacando las lecciones aprendidas y las medidas de seguridad recomendadas. Se espera que este documento sirva como un recurso valioso para comprender y abordar las amenazas comunes en el ámbito de la seguridad web.

# Informe Ejecutivo

## a. Breve resumen del proceso realizado

Durante el proceso de auditoría de seguridad en la aplicación WebGoat, se establecieron e hicieron varias fases para evaluar la resistencia de la aplicación a posibles amenazas según algunas de las vulnerabilidades más comunes del OWASP Top 10. Inicialmente, se realizó un reconocimiento para identificar posibles puntos de entrada y recopilar información sobre el sistema operativo y servicios. Después se procedió a la fase de explotación, donde se aplicaron distintas técnicas. Tras la identificación de estas vulnerabilidades, siguió a la fase de post-explotación, evaluando la posibilidad de acceder a información sensible y comprometer la integridad de la aplicación.

## b. Vulnerabilidades destacadas

Durante la auditoría, se identificaron varias vulnerabilidades críticas que podrían comprometer la seguridad de la aplicación. Entre las más destacadas se incluyen:

* SQL Injection (A1 - OWASP Top 10): Se encontraron instancias de inyección SQL, permitiendo a un atacante ejecutar consultas maliciosas en la base de datos subyacente.
* Cross-Site Scripting (A3 - OWASP Top 10): Se detectaron vulnerabilidades XSS que podrían permitir a un atacante inyectar scripts maliciosos en las páginas web visitadas por otros usuarios.
* XML External Entity (XXE) Injection (A4 - OWASP Top 10): Se identificaron puntos débiles en la gestión de datos XML que podrían ser explotados para realizar ataques de inyección XXE, comprometiendo la confidencialidad de la información.

## c. Conclusiones

La aplicación WebGoat exhibe múltiples vulnerabilidades que podrían ser explotadas por actores maliciosos. Estas debilidades, si no se abordan de manera inmediata, podrían poner en riesgo la confidencialidad, integridad y disponibilidad de los datos manejados por la aplicación.

## d. Recomendaciones

Es recomendable implementar medidas de seguridad adicionales, como la validación adecuada de datos de entrada, el uso de parámetros preparados para consultas SQL y la implementación de filtros para prevenir ataques XSS. Además, se aconseja una revisión exhaustiva de la gestión de XML para evitar vulnerabilidades XXE.

# Descripción del Proceso de Auditoría

## a. Reconocimiento/Information gathering

En esta fase, se llevó a cabo un análisis de la aplicación WebGoat para recopilar información sobre su arquitectura, tecnologías utilizadas y posibles puntos de entrada. Se utilizaron herramientas como Nmap y enumeración manual para identificar servicios en ejecución, puertos abiertos y posibles vulnerabilidades.

A computer screen shot of a computer program

Description automatically generated

Aquí se aprecia un escaneo de posibles puertos abiertos utilizando Nmap. Los puertos 8080, 8081 y 9090 están abiertos (los puertos 37522 y 39427 también, pero son “unknown”, desconocidos).

Texto

Descripción generada automáticamente

Seguimos con un escaneo del sistema operativo de la máquina en el que WebGoat está instalado. En este caso, es Linux 2.6.32, el cual sabemos que es nuestro propio sistema operativa dentro de nuestra maquina virtual.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Nmap has buscado los servicios asociados a los puertos abiertos de WebGoat 8080, 8081 y 9090. Ha encontrado un servicio por puerto. El servicio asociado al puerto 8080 está relacionado con el http proxy, lo cual es lógico por el hecho de que se está utilizando esta máquina para cargar WebGoat.

## b. Explotación de vulnerabilidades detectadas

Durante la fase de explotación, se identificaron y evaluaron varias vulnerabilidades en la aplicación. La inyección SQL se llevó a cabo mediante la manipulación de datos de entrada, lo que permitió la extracción de información sensible de la base de datos. Además, se realizaron ataques XSS exitosos, demostrando la posibilidad de ejecutar scripts maliciosos en el contexto del navegador del usuario.

Patrón de fondo

Descripción generada automáticamente

Inyección SQL. Esta base de datos es vulnerable a una inyección numérica SQL relativamente sencilla. Se prueba ingresando “'1' OR '1'='1' –“ en los dos campos para saber cual de los dos campos es vulnerable. Esta es una inyección clásica que sirve para que el query resulte en un VERDADERO, y que por lo tanto muestre todos los resultados.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

También por lógica sería evidente que el login\_count es un valor numérico mientras que el user\_id podría ser un string o también un valor numérico. Con esta prueba, confirmamos que login\_count es numérico y por lo tanto no es el campo vulnerable ya que no admite OR en el campo. Ponemos un valor numérico en login\_count y la inyección en el campo user\_id.

Forma, Rectángulo

Descripción generada automáticamente

El resultado es que tenemos acceso visual a todos los datos de la tabla.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Aquí vemos que al hacer VERDADERO la consulta, nos da todos los resultados. La consulta SELECCIONA (SELECT) todos (\*) de la tabla (user\_data) con la condición (WHERE) de que la columna Login\_Count sea igual a 1 (que como vemos no lo es, pero poco importa) O (or) que sea 1 = 1, que es la forma de decir que salga TRUE, VERDADERO. O se cumple una condición o la otra. Mientras se cumpla como mínimo UNA de las dos, ejecutará la consulta y nos mostrará todos los resultados.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Inyección string SQL. Esta base de datos también es vulnerable a una inyección SQL, en este caso con una concatenado a un campo string, representado por Lastname. Esta vulnerabilidad permite mostrar todos los datos confidenciales que se guardan, y es otra vez bastante sencillo de hacer. Sabemos nuestro propia autenticación TAN, pero no sabemos el de los otros trabajadores. Sin embargo veremos que es fácil averiguarlo y además conocer algo tan privado como los salarios de los otros empleados.

Interfaz de usuario gráfica

Descripción generada automáticamente

Como ya sabemos que el campo de employee name es un campo string y que la consulta buscaría el apellido del empleado, le añadimos esta inyección a la consulta que concatenará este código a la consulta. Aunque no ponemos un apellido en concreto, no hace falta. La consulta se hace aún que esté vacío. O bien busca el apellido en vacío (“ “) o (OR) 1=1 el cual es como antes TRUE, VERDADERO. En consecuencia, nos mostrará los datos de todos los empleados como se muestra en la siguiente imagen.

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

Igual que antes, es un problema serio el que pueda ver datos tan sensibles como estos de otros empleados.

Interfaz de usuario gráfica

Descripción generada automáticamente

Aquí vemos otra posible vulnerabilidad, entrando en el terreno de las alertas en un entorno web HTML. Este carrito donde se añaden distintos productos para comprar puede tener un fallo, y se puede comprobar mediante un ‘alert()’ o un console.log(), en lo que se denomina como un ataque cross-site scripting. Es decir, utilizar la programación JavaScript para causar un comportamiento no deseado en una aplicación web. Aquí le añadimos un alert() a uno de los campos.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Y el resultado, con el siguiente código “<script>alert("oh no I am very vulnerable!");</script>”, el cual abre una ventana nueva con un mensaje específico, que nos da es el siguiente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Aunque el ataque no haya causado un daño como el que puede tener una base de datos al mostrar todos los usuarios y su información, puede tener otros usos dañinos. Estos scripts alteran el comportamiento de ciertas páginas, pudiendo redirigir a usuarios hacia páginas web falsas (phishing) o incluso robar datos privados del usuario.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En este ejemplo, podemos intentar aprovechar la vulnerabilidad en cómo se procesa el XML para descubrir el contenido de un directorio. Utilizamos burpsuite para capturar la carga de la página.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Al darle a SUBMIT en la página de WebGoat, burpsuite recupera el POST, y ya nos está dando algo de información sobre el procesamiento XML.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Nos da la versión y el código HTML de cómo se ha construido está página. A continuación, vamos a mandar esta información al repetidor.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Ahora cambiamos el código HTML para incluir el directorio para mostrarnos lo que contiene.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Y vemos que nos da el resultado correcto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Con esto ya sabemos lo que contiene el directorio de la máquina y algunos de sus archivos en file///.

También el mismo ataque se puede decir aunque se esté utilizando un tipo de dato distinto al XML. Por ejemplo, si fuera un JSON en vez de XML sería muy parecido.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Escribimos un comentario, el cual interceptamos con burpsuite.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Desde aquí cambiamos el tipo de dato de un JSON a un XML. Texto

Descripción generada automáticamente

Y procedemos a volver a inyectar en el código HTML lo que nos revelará el contenido del directorio.

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Vemos como al igual que antes ha aceptado el cambio de tipo de dato junto con la alteración al procesamiento del XML, mostrándonos los contenidos del directorio.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Este es un ejemplo de un query vulnerable y no vulnerable. En WebGoat se especifica que el de arriba debería saltar una alerta de javascript, es decir una ventana que se abre diciendo “XSS”. Sin embargo al darle a Go! no salta ningún mensaje. El mensaje de abajo sin embargo si salta.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En este apartado, se pide utilizar una contraseña que se lo suficientemente “fuerte” para alcanzar los 4 criterios que normalmente se pide en cualquier aplicación donde se registre un usuario. Probamos “password” como contraseña.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

La contraseña “password” es un clásico de lo obvio que es. Según WebGoat literalmente no se tardaría ni un segundo en adivinarlo o averiguarlo con un programa que craquee contraseñas. Ahora para comparar probamos la contraseña “Pwease@Cutdatomato61”.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

La diferencia es que se han utilizado una combinación de símbolos, mayúsculas, letras y números, que, aunque formen ciertas palabras que son familiares, su combinación no es común. Estima que, para averiguar esta contraseña, se tardarían 4050608828 años por un programa.

## c. Post-explotación

Después de explotar las vulnerabilidades, podemos ver lo crítico que es proteger ante todo la confidencialidad de nuestros datos. Estos fallos nos revelan la importancia de los tres principios de la triada: CIA (Confidentiality, Integrity, Availability). Se logró acceder a datos confidenciales de la base de datos, lo que destaca que se ha roto la confidencialidad de los datos, incluyendo datos sensibles como los salarios, números de teléfono, nombres, apellidos, departamentos donde trabajan los empleados, etc. Estos fallos también son un ejemplo de la falta de integridad, ya que es posible también alterar los datos sin haber ningún tipo de backup o forma de corroborar la validez o integridad de los datos, al mismo tiempo que podemos darnos o quitar permisos a los usuarios o administradores. Aunque no es obvio, también vemos que falla en el principio de disponibilidad, porque el hecho es que estos fallos se pueden evitar si se mantiene un entorno actualizado y con un sistema operativo que esté actualizado.

## d. Posibles mitigaciones

Para la inyección SQL:

* Es necesario que desde el código que se utilice se implementen consultas parametrizadas en lugar de cadenas SQL concatenadas.
* Utilizar formas de validar los datos que entran en el código antes de que se utilicen en la consulta SQL.
* Proporcionar y limitar los usuarios que pueden manipular la base de datos, dando o restringiendo los privilegios de usuario.

Para los ataques Cross-site Scripting:

* Utilizar una codificación de salida, es decir el Output Encoding, para el lado cliente.
* Filtras los datos de entrada desde el mismo servidor para evitar cualquier script malicioso.

Para la inyección XML External Entity:

* Se recomienda deshabilitar el procesamiento de entidades externas, para que no utilice archivos remotos o locales.

## e. Herramientas utilizadas

Se emplearon varias herramientas, entre las que se incluyen Nmap para el escaneo de puertos, Burp Suite para la manipulación de datos de entrada y la identificación de vulnerabilidades XML, y SQLMap para la exploración y explotación de posibles inyecciones SQL.