Ciberseguridad Práctica 3

**Escaneo de Red**

Daniel Varela Sánchez & Guillermo Martín-Coello Juárez

# 

# **Introducción**

La seguridad web es esencial para el correcto funcionamiento de todos los servicios que se desarrollan a través de internet. En esta práctica, el objetivo es aprender y comprender habilidades de seguridad ofensiva, para tener conseguir perspectiva de lo fácil y vulnerable que es el software mal protegido. El escenario a explorar consiste en una web vulnerable. La web vulnerable que se explorará es una representación de muchos sitios web reales que presentan vulnerabilidades de seguridad comunes. Al comprender las vulnerabilidades en la seguridad de la web, los profesionales de la ciberseguridad pueden desarrollar estrategias para proteger sus sistemas y aplicaciones de los ataques maliciosos.

Es importante mencionar que el objetivo de explorar una web vulnerable no es fomentar la actividad ilegal o malintencionada. En cambio, se trata de una práctica ética que se utiliza para mejorar la seguridad y protección de los sistemas y aplicaciones. La seguridad es un esfuerzo constante y en constante evolución, y los profesionales de la ciberseguridad deben estar capacitados y preparados para detectar y prevenir ataques maliciosos.

# 

# **Desarrollo**

# **Ataques de inyección HTML reflejado (1 pto.)**

La inyección de código HTML se produce cuando un atacante logra introducir código malicioso en una página web a través de una entrada de datos no controlada, como un formulario de comentarios, una barra de búsqueda o una URL. Una vez que se inserta el código malicioso, se ejecuta en el navegador web del usuario que accede a la página web, lo que puede permitir al atacante robar información, tomar el control de la sesión de usuario o incluso infectar el dispositivo del usuario con malware.

En nuestro caso hemos atacado la vulnerabilidad del html editando la url del inicio de sesión para que envíe una imagen, con el siguiente código:

| <http://192.168.56.101/bWAPP/htmli_get.php?firstname=%3Cimg%20src=%22https://images-ext-1.discordapp.net/external/lyYElIuFOVebKYNNeqzNNhUJ6XQWCnBl6XVDrmBrLRI/https/cdn.memegenerator.es/imagenes/memes/full/4/44/4448349.jpg%22%20style=%22height:150px%22%3E&lastname=%3Cimg%20src=%22https://www.eurotopics.net/cache/images/0/182880-5x3-topteaser1260x756.jpg?9AE6E%22%20style=%22height:150px%22%3E&form=submit> |
| --- |
|  |
| Imagen 1.1: HTML Injection - Reflected (GET) |

Éste caso es solo una prueba inofensiva, pero los ataques de inyección de código HTML pueden tener graves consecuencias como el robo de información o la propagación de malware. Los desarrolladores web deben encargarse de que las entradas de datos html se validen y sean correctas para poder evitar este tipo de ataque.

Otra manera de atacar a través de inyección html es modificando los parámetros de una petición de tipo POST. Hemos replicado este posible ataque utilizando la herramienta Burp:

|  |
| --- |
|  |
| Imágenes 1.2: HTML Injection - Reflected (POST) |

Como se puede observar en las Imágenes 1.2, hemos cambiado el formulario POST para incluir inyección de html, dando como resultado la imagen final, con el texto que hemos añadido en ella.

Finalmente, el ataque por edición de URL implica que la víctima accede a una dirección diferente a la correcta. Para replicarlo hemos utilizado la herramienta Burp para editar la URL. Ésto se puede observar en la siguiente imágen:

|  |
| --- |
| Imagen 1.3: HTML Injection - Reflected (URL) |

Como se puede observar, utilizamos Burp para cambiar la URL por una maliciosa.

**Ataques de inyección SQL:**

Los ataques de inyección SQL son una técnica malintencionada utilizada por los atacantes para insertar código malicioso en una consulta SQL y así obtener acceso no autorizado a información almacenada en una base de datos vulnerable.

En el caso de los ataques de inyección SQL simples (GET/Search), los atacantes aprovechan las entradas de búsqueda en formularios web para introducir código SQL malicioso. Cuando el servidor web procesa la consulta de búsqueda, el código malicioso se ejecuta y elimina, edita u obtiene los datos vulnerables. Hemos accedido con inyección SQL simple utilizando el siguiente código:

| ññññ’ union select 1,2,3,4,5,6,7 from users # |
| --- |
|  |
| Imagen 2.1: SQL Injection - (GET/Search) 1 |

Como se puede observar, en la imagen 2.1 introducimos un código SQL malicioso, con caracteres poco habituales al inicio para no obtener resultados de búsqueda, junto con un UNION para hacer un reconocimiento de las columnas y la estructura de la tabla.

| ññññ’ union select "",login,password,"",email,"","" from users # |
| --- |
|  |
| Imagen 2.2: SQL Injection - (GET/Search) 2 |

Finalmente, en la Imagen 2.2 insertamos un comando para obtener los datos vulnerables. Así, escogiendo las columnas deseadas, podemos acceder a datos vulnerables, como en éste caso los emails y las contraseñas (cifradas).

En el caso de los ataques de inyección SQL en base de datos (SQLite), se pueden explotar vulnerabilidades en sistemas de gestión de bases de datos (DBMS) para ejecutar código SQL malicioso. Para realizarlo, hemos utilizado el siguiente código:

| ñññ' UNION ALL SELECT 1,sqlite\_version(),1,1,1,1; |
| --- |
|  |
| Imagen 2.2: SQL Injection - (SQLite) |

Como se puede observar en la imagen 2.2, hemos añadido la llamada a una función al gestor de base de datos, en este caso para preguntar por la versión del gestor. Con fines maliciosos esto permitiría atacar la base de datos.

En los ataques de inyección SQL almacenados (Blog), los atacantes pueden explotar vulnerabilidades en las aplicaciones web que permiten a los usuarios introducir información en una base de datos. Hemos vulnerado el sistema con un ataque de inyección SQL almacenado utilizando el siguiente código:

| sda',(select password from mysql.user where user='root' limit 0,1))# |
| --- |
|  |
| Imagen 2.3: SQL Injection - Stored (Blog) |

Como se puede observar en la imagen 2.3, hemos utilizado la estructura que utiliza la base de datos para enviar los comentarios (comentario, usuario) y hemos inyectado código SQL malicioso para que nos devuelva la contraseña del usuario root (Eso sí, cifrada).

Para prevenir los ataques de inyección SQL, es importante que los desarrolladores web utilicen técnicas seguras de codificación, como la validación de entradas y la sanitización de datos. Además, el uso de parámetros preparados en lugar de la concatenación de cadenas de SQL puede ayudar a prevenir los ataques de inyección SQL.

**Autenticación (4 ptos.)**

La autenticación es el proceso de verificar la identidad de un usuario antes de permitir el acceso a un sistema o aplicación. Hemos atacado varias de las posibles vulnerabilidades del proceso de autenticación:

Los CAPTCHA pueden ser efectivos para prevenir ataques automatizados, pero los atacantes pueden utilizar esquivar estas protecciones CAPTCHA para superar esta medida de seguridad. Para probarlo, hemos atacado el uso de captcha denegando su uso, utilizando la herramienta Burp. Hemos activado la intercepción, dónde podemos decidir aceptar el envío de la página a la que queremos acceder, pero denegar el envío del php (captcha\_box.php). Se puede observar en las siguientes imágenes (3.1)

|  |
| --- |
| Imágenes 3.1: CAPTCHA |

La gestión de sesiones también es un aspecto crítico de la autenticación. Es importante evitar el uso de ID de sesión en la URL, ya que esto puede permitir que un atacante intercepte la sesión de un usuario y acceda al sistema en su lugar. Hemos reproducido el ataque, como se puede observar en las siguientes imágenes:

|  |
| --- |
| Imágenes 3.2: Session Management - Session ID in URL |

Como se puede ver en las imágenes 3.2, creamos un nuevo usuario para entrar en el sistema. Una vez hecho esto, cambiamos la id de sesión por otra para acceder a la sesión de otro usuario esquivando los requerimientos de autenticación.

Las cookies, para ser seguras, deben estar protegidas con el atributo HTTPOnly. Cuando se establece este atributo, la cookie sólo es accesible a través del protocolo HTTP y no a través de scripts del lado del cliente, lo que evita que los atacantes accedan a la cookie a través de ataques de cross-site scripting (XSS). Hemos atacado esta vulnerabilidad cambiando la ID de sesión como se muestra en las siguientes imágenes (3.3):

|  |
| --- |
| Imagen 3.3: Session Management - Cookies (HTTPOnly) |

Otra técnica es la exploración de directorios y archivos. Los atacantes pueden intentar acceder a directorios y archivos en un servidor web que no deberían estar públicamente disponibles. Hemos accedido a archivos que deberían ser inaccesibles, simplemente cambiando la url para movernos de directorio, como se muestra en las siguientes imágenes (3.4).

|  |
| --- |
| Imagen 3.3: Session Management - Directory Traversal - Directories y Files |

**Cross-Site-Scripting**

El Cross-Site Scripting (XSS) es una vulnerabilidad web que permite a un atacante inyectar código malicioso en una página web visitada por un usuario. Esto puede llevar a una variedad de ataques, incluyendo el robo de información del usuario, el control de su sesión en el sitio web, y la distribución de malware.

La vulnerabilidad de Reflejado (GET) se produce cuando una aplicación web no valida correctamente la entrada del usuario en una solicitud GET, permitiendo que un atacante inyecte código malicioso en la respuesta que se devuelve al usuario. Para replicar este ataque, utilizamos la vulnerabilidad de la página web en la solicitud GET, insertando código malicioso. Esto se puede observar en las siguientes imágenes, y el siguiente código:

| **<script>alert(document.cookie)</script>** |
| --- |
|  |
| Imágenes 4.1: Cross-Site-Scripting - Reflejado (GET) |

Las imágenes 4.1 muestran cómo inyectamos código javascript en un formulario para encontrar la cookie con la ID de sesión.

**Sensitive Data Exposure**

La Sensitive Data Exposure es una vulnerabilidad que se produce cuando los datos vulnerables se almacenan o transmiten sin protección suficiente. La codificación Base64 es una forma común de codificar datos binarios en texto ASCII, lo que permite que los datos se transmitan a través de medios que solo admiten caracteres de texto, como el correo electrónico o los formularios web. Sin embargo, la codificación Base64 no proporciona cifrado o protección adicional para los datos confidenciales. En realidad, es una forma de "ocultar" los datos, pero no de cifrarlos, y por lo tanto no son seguros.

Si un atacante intercepta los datos codificados en Base64, puede fácilmente decodificarlos para revelar la información confidencial en texto claro. Por lo tanto, la codificación Base64 no es adecuada para proteger datos confidenciales en reposo o en tránsito.

|  |
| --- |
| Imágenes 5.1: Sensitive Data Exposure - Codificación B64 |

En las imágenes (5.1) se puede observar cómo obtenemos el valor cifrado “secret” de la cookie, y lo desciframos sin problema utilizando Base64 para encontrar el mensaje sin cifrar.

Para protegerse contra la Sensitive Data Exposure en la codificación Base64, se cifrar los datos confidenciales antes de codificarlos en Base64. Por ejemplo, se puede utilizar una capa de conexión segura (SSL/TLS) para cifrar los datos durante la transmisión y luego codificarlos en Base64 para su transporte. Otra opción es utilizar una función de hash segura para cifrar la información confidencial antes de codificar en Base64.

# 

# **Conclusiones**

Tras la realización de éste ejercicio, hemos ampliado nuestra comprensión de la importancia de la seguridad web y las vulnerabilidades comunes que pueden ser explotadas por los hackers para realizar ataques maliciosos. La seguridad web es un esfuerzo constante y en constante evolución, y los profesionales de la ciberseguridad deben estar capacitados y preparados para detectar y prevenir cualquier tipo de ataque malicioso.

Hemos explorado múltiples formas de ataque, como los ataques de inyección HTML reflejado y los ataques de inyección SQL. También se exponen los ataques de autenticación y de evitar CAPTCHA, manipular la gestión de sesiones, explotar vulnerabilidades en las cookies y utilizar la técnica de Directory Traversal para acceder a archivos y directorios no autorizados.

Por último, se menciona la vulnerabilidad de Sensitive Data Exposure y se destaca la importancia de utilizar la codificación adecuada, como B64, para proteger la información sensible.

La mayoría de vulnerabilidades, entre otras muchas, son fácilmente corregibles si se tiene en cuenta la seguridad como un elemento de diseño a la hora de desarrollar el software, y no se trate de “arreglar” una vez construido. En general, la práctica busca concienciar sobre la importancia de la seguridad web y la necesidad de estar siempre alerta y actualizado para proteger los sistemas y aplicaciones de posibles ataques maliciosos.