Laboratorio de ejercicios de vulnerabilidades de aplicaciones web

Tema 4. Detección y corrección de vulnerabilidades de aplicaciones web

Índice general

Capítulo 1

SQL Injection

Pese a que hicimos hincapié en esta vulnerabilidad durante las sesiones teóricas, vamos a realizar otro ejemplo: emplearemos SQLMap para automatizar una inyección ciega (muy tediosa de forma manual).

Para realizar el ataque vamos a utilizar la aplicación DVWA, en concreto su sección *SQL Injection (Blind)*. Antes de utilizar SQLMap necesitamos cierta información, ya que se trata de una aplicación web que requiere identificarse como usuario.

¿Qué podemos necesitar para emplear SQLMap en estas circunstancias? ¿Cómo podemos obtener esta información?

Como seguramente habéis pensado, necesitamos una sesión activa para poder entrar a la aplicación web, es decir, un token de sesión. Lo mismo necesitará SQLMap para realizar la inyección. Vamos a utilizar la herramienta Burp Suite (preinstalada en Kali Linux) para obtener el token y otras cookies necesarias (imaginad que no tuviéramos otra forma de obtener esta información).

En primer lugar, ejecutamos Burp Suite en Kali Linux con todas las opciones de configuración inicial por defecto. En la parte inferior de su interfaz, Burp Suite nos muestra dónde está ejecutando el proxy (Figura 1.1).

Figura 1.1: Dirección del proxy de Burp Suite.

Posteriormente, configuramos el modo proxy en nuestro navegador (Figura 1.2). A partir de ahora, las peticiones que hagamos a DVWA pasarán primero por Burp Suite.

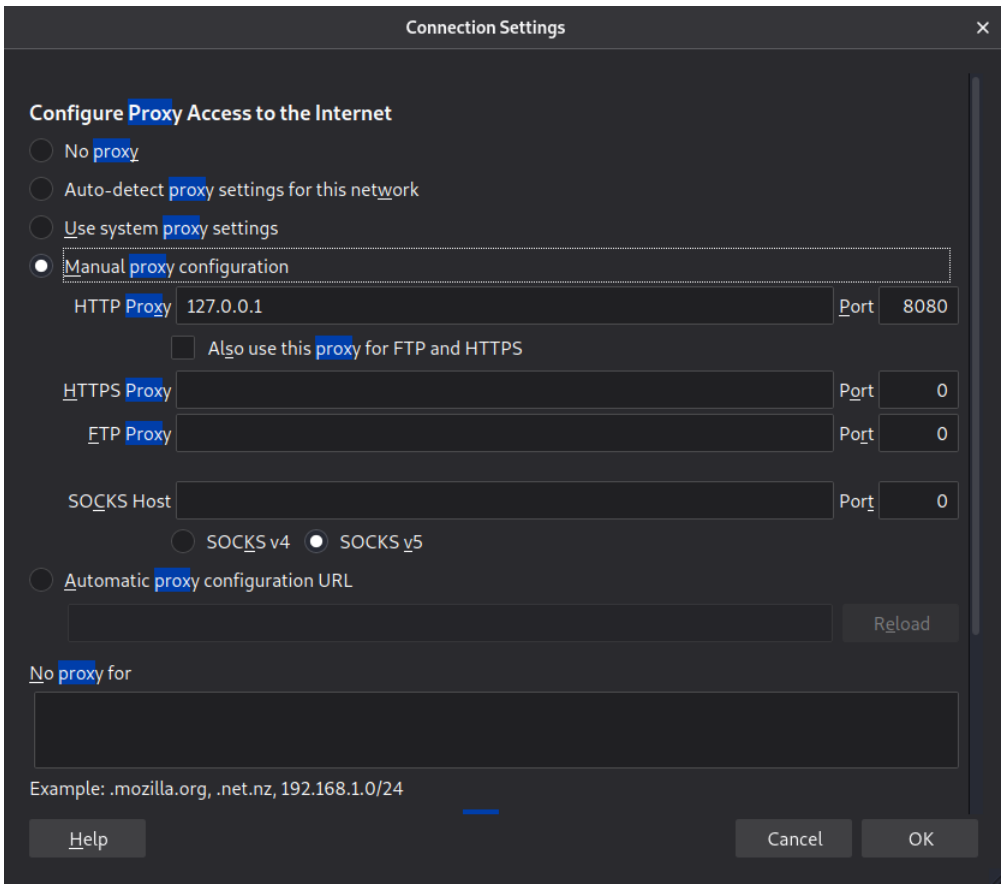

Figura 1.2: Configuración del proxy en Firefox.

Finalmente, empleamos Burp Suite para capturar una petición a la aplicación web que vamos a 'atacar'. En el apartado superior de la interfaz de Burp Suite encontramos la pestaña 'Proxy'. En esta podemos ver todas las peticiones que hagamos desde nuestro navegador, reenviarlas e incluso descartarlas. Entramos a DVWA y vemos como nuestro navegador web se queda a la espera de reenviar o descartar la petición desde Burp Suite. Si nos fijamos en la petición, vemos las cookies que necesitamos para poder usar SQLMap, en concreto 'security' y 'PHPSESSID' (Figura 1.3).

NOTA: recordad iniciar sesión con usuario y contraseña en DVWA para enlazar el token de sesión con una entidad en la aplicación web (es decir, con el usuario 'admin').

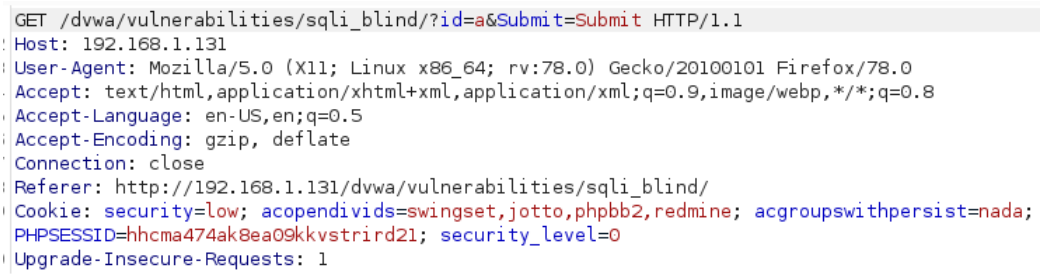

Figura 1.3: Captura de petición con Burp Suite.

Si nos fijamos en la cookie PHPSESSID antes y después de iniciar sesión, ¿notamos algo extraño? ¡¡El token de sesión es el mismo antes y después de autenticarnos!!

Una vez que ya tenemos las cookies necesarias, empezamos con el ataque de inyección a ciegas en SQLMap. En cuanto a la información que queremos obtener, primero serán las bases de datos y las tablas de la base de datos objetivo. Tras esto, obtenemos las columnas de la tabla a atacar, y por último, los datos de las columnas que queramos de dicha tabla.

El objetivo es obtener usuarios y contraseñas (aunque estén cifradas) de la tabla *users* de la base de datos *dvwa*.

Con los siguientes comandos obtenemos información de la base de datos con SQLMap. Guardad la información obtenida para los siguientes ejercicios.

NOTA 1: con el comando *sqlmap –help* podéis ver qué significan cada una de las flags especificadas.

NOTA 2: en el último comando responder no (n) a las preguntas que se os hagan.

Listing 1.1: Obtención de bases de datos.

```
sqlmap -u "http://192.168.0.50/dvwa/vulnerabilities/sqli_blind/?id=1&
    Submit=Submit#" --cookie="security=low;PHPSESSID=
    hhcma474ak8ea09kkvstrird21;" --dbs
```

Listing 1.2: Obtención de tablas.

```
sqlmap -u "http://192.168.0.50/dvwa/vulnerabilities/sqli_blind/?id=1&
    Submit=Submit#" --cookie="security=low;PHPSESSID=
    hhcma474ak8ea09kkvstrird21;" -D dvwa --tables
```

Listing 1.3: Obtención de columnas.

```
sqlmap -u "http://192.168.0.50/dvwa/vulnerabilities/sqli_blind/?id=1&
    Submit=Submit#" --cookie="security=low;PHPSESSID=
    hhcma474ak8ea09kkvstrird21;" -D dvwa -T users --columns
```

Listing 1.4: Obtención de usuarios y contraseñas.

```
sqlmap -u "http://192.168.0.50/dvwa/vulnerabilities/sqli_blind/?id=1&
    Submit=Submit#" --cookie="security=low;PHPSESSID=
    hhcma474ak8ea09kkvstrird21;" -D dvwa -T users -C user,password --
    dump
```

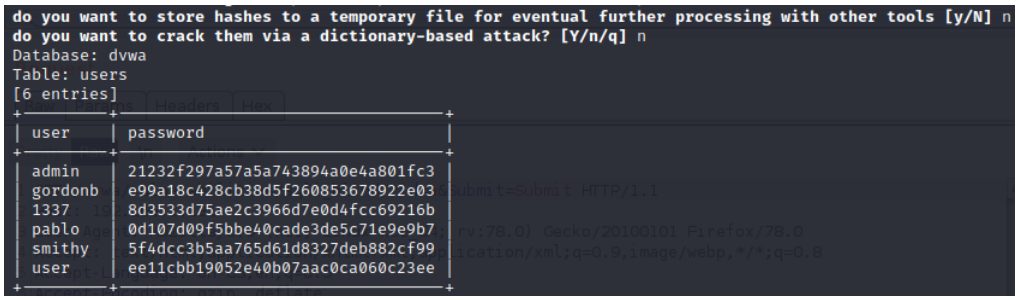

Figura 1.4: Resultado que debemos obtener con SQLMap.

Capítulo 2

Ataques a credenciales

En el ejercicio anterior hemos obtenido los usuarios y las contraseñas de DVWA. Pero como hemos visto, estas contraseñas suelen (deberían) guardarse cifradas.

En este capítulo vamos a tratar de romper el cifrado de dichas contraseñas (ataque offline). Además, también realizaremos un ataque de diccionario contra formularios (ataque online).

2.1. Ataque offline

En primer lugar vamos a tratar de obtener las contraseñas en claro del ejercicio anterior. Para ello, utilizaremos la famosa herramienta de password cracking *John the Ripper* junto con *hash-identifier* para identificar el tipo de hash al que nos enfrentamos (las dos preinstaladas en Kali Linux).

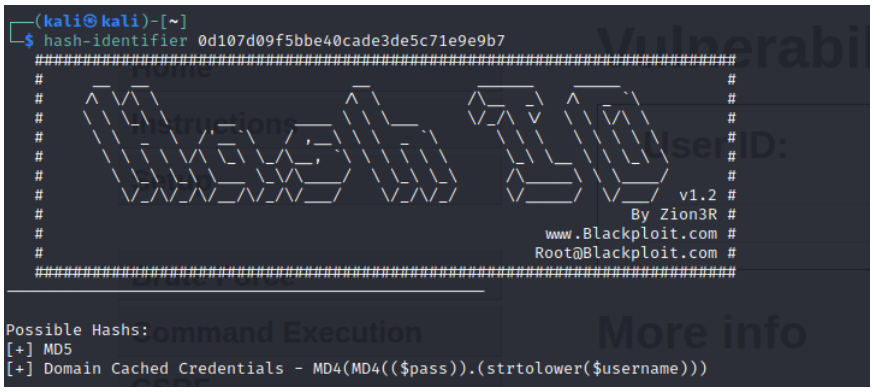

Figura 2.1: Identificamos el tipo del hash.

Nos centraremos en el hash del usuario *pablo*. Esta vez emplearemos el

famoso diccionario *rockyou*, un diccionario muy común en desafíos y retos de seguridad donde se premia el identificar el posible ataque por diccionario, y no tanto la creación de un diccionario como tal. El primer paso, como hemos dicho, es identificar el tipo de hash (Figura 2.1):

Listing 2.1: Identificamos el hash.

```
$ hash-identifier 0d107d09f5bbe40cade3de5c71e9e9b7
```

Ahora que sabemos de qué tipo de hash se trata podemos emplear John the Ripper, pero antes necesitamos descomprimir rockyou, ya que el diccionario se encuentra comprimido por defecto.

Listing 2.2: Descomprimimos rockyou.

```
# sudo gzip -d /usr/share/wordlists/rockyou.txt.gz
```

Ahora que conocemos el tipo de hash y tenemos un diccionario relativamente potente... **toca construir el comando.** Pistas:

- Utilizad el comando *john - -help* para revisar las posibles opciones a configurar.
- Es posible que queráis guardar el hash en un fichero...
- Es importante indicar el tipo de hash que queremos crackear (Raw-MD5).

¿Cuál es la contraseña del usuario *pablo*? ¿Podéis repetir el proceso con el usuario *admin*?

2.2. Ataque online

En este caso vamos a atacar directamente el formulario de inicio de sesión de Mutillidae II, pero no vamos a utilizar los diccionarios por defecto, en su lugar vamos a generar uno propio. Para esto emplearemos la herramienta *Cewl*, la cual permite generar diccionarios directamente desde páginas web mediante URL. **Si sabemos que a nuestra víctima le gusta el fútbol, podríamos utilizar un diario deportivo para tratar de adivinar la contraseña (¿marca.com?)**.

En este caso emplearemos la web *bab.la*. Esta página permite traducir de inglés a español, pero también muestra texto donde aparece una palabra concreta. De esta forma, vamos a generar un diccionario para atacar al usuario *admin*, empleando la palabra *admin* en el buscador de bab.la.

Listing 2.3: Creando diccionario con Cewl.

```
$ cewl https://es.bab.la/diccionario/ingles-espanol/admin -d 1 -m 5 -
    w diccionario.txt
```

Como veis, en el comando indicamos la URL de la búsqueda de bab.la. Los siguientes parámetros indican:

- -d: la profundidad de la búsqueda. Es importante que sea 1 para evitar que nos metan timeouts, si fuese superior el programa buscaría palabras dentro de los enlaces también.
- -m: establece el tamaño mínimo de las palabras para ser incluidas en el diccionario.
- -w: escribe el diccionario generado en la ruta que le indicamos (nombre del diccionario).

Con esto ya tenemos nuestro diccionario personalizado. Ahora vamos a tratar de identificarnos en Mutillidae II (Figura 2.2).

Figura 2.2: Login Mutillidae II.

En este ejercicio solo vamos a atacar la contraseña del usuario *admin* (sabemos que el usuario existe ya que cuando introducimos el usuario y una contraseña incorrecta deliberadamente, el mensaje que obtenemos es 'Password incorrect'). **¡MENSAJE VULNERABLE!**

Configuramos el proxy en nuestro navegador como en el primer ejercicio y abrimos Burp Suite. Ahora intentamos iniciar sesión con el usuario *admin* y una contraseña aleatoria, y capturamos la petición en Burp Suite. Posteriormente, como vemos en la Figura 2.3, en acciones elegimos *Send to intruder* (herramienta/plugin de automatización de ataques en Burp Suite). Si hemos pulsado forward y no tenemos la petición retenida, podemos hacer lo mismo desde la pestaña *HTTP history* al lado de la pestaña *Intercept*.

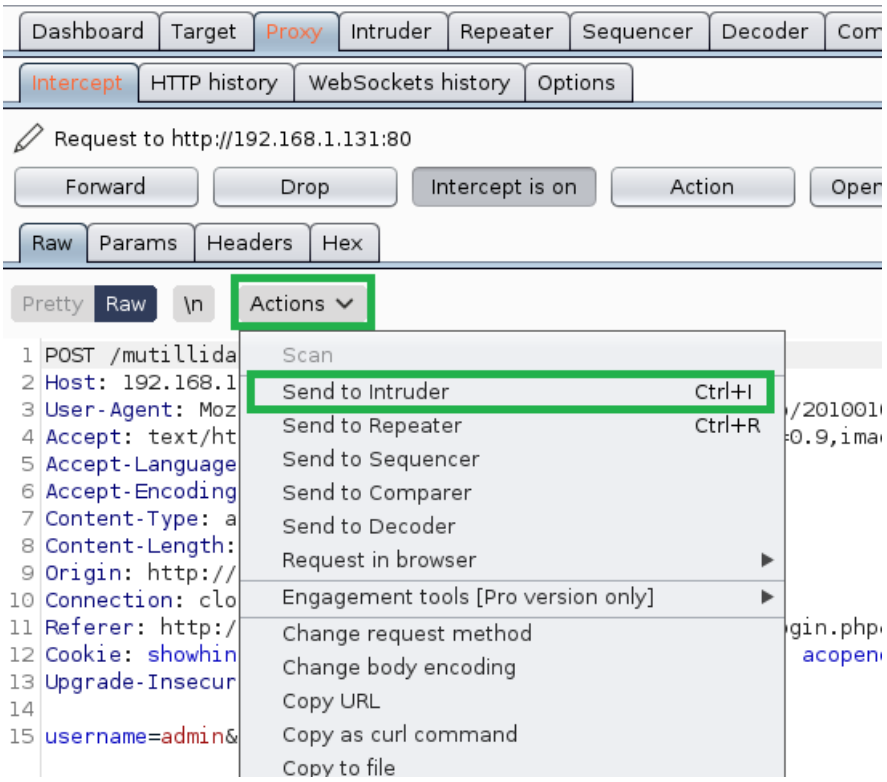

Figura 2.3: Enviamos la petición al Intruder.

Una vez que hemos enviado la petición al Intruder, podemos reenviar o descartar la petición y dejar de interceptar otras ('Intercept is on'). Para realizar el ataque, vamos a la pestaña del Intruder (Figura 2.4), subpestaña 'Positions' y pulsamos el botón 'Clear' en la parte derecha de la interfaz de Burp Suite (si no hacemos esto, intentará atacar todos los parámetros de la petición marcados en verde). Después, seleccionamos con el ratón (arrastrando) la parte con la contraseña aleatoria que introdujimos antes y pulsamos el botón 'Add' (también a la derecha, encima de 'Clear'). Una vez hecho esto solo debería estar marcada la contraseña como parámetro a atacar (Figura 2.5).

Figura 2.4: Enviamos la petición al Intruder.

```
username=admin&password=§a§&login-php-submit-button=Login
```

Figura 2.5: Elemento password seleccionado.

Habiendo seleccionado el campo a atacar, vamos a la subpestaña 'Payloads'. En 'Payload Options' elegimos el diccionario que creamos antes con *Cewl* (no os preocupéis si algunas palabras tienen caracteres extraños, las 'ñ' y los acentos las provocan). Podemos ver cómo queda la configuración en la Figura 2.6. Una vez que tenemos el diccionario cargado, pulsamos 'Start attack' (arriba a la derecha en la interfaz de Burp Suite).

Finalmente, se abre una nueva ventana con cada una de las palabras probadas como contraseña (Figura 2.7). Al cabo de un rato, si ordenamos la columna 'Status' en orden descendente, vemos un código HTTP 302 Found diferente a las demás pruebas (además que la longitud de la respuesta también es diferente). **Hemos encontrado la contraseña: 'admin'**.

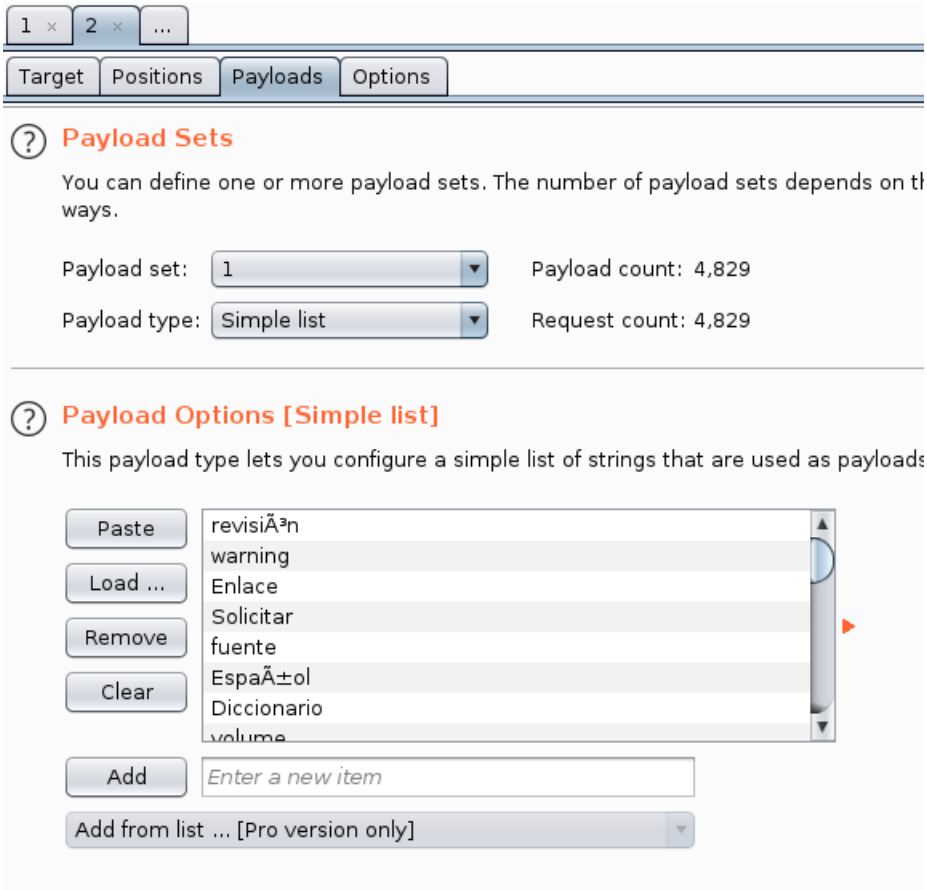

Figura 2.6: Elección de payload.

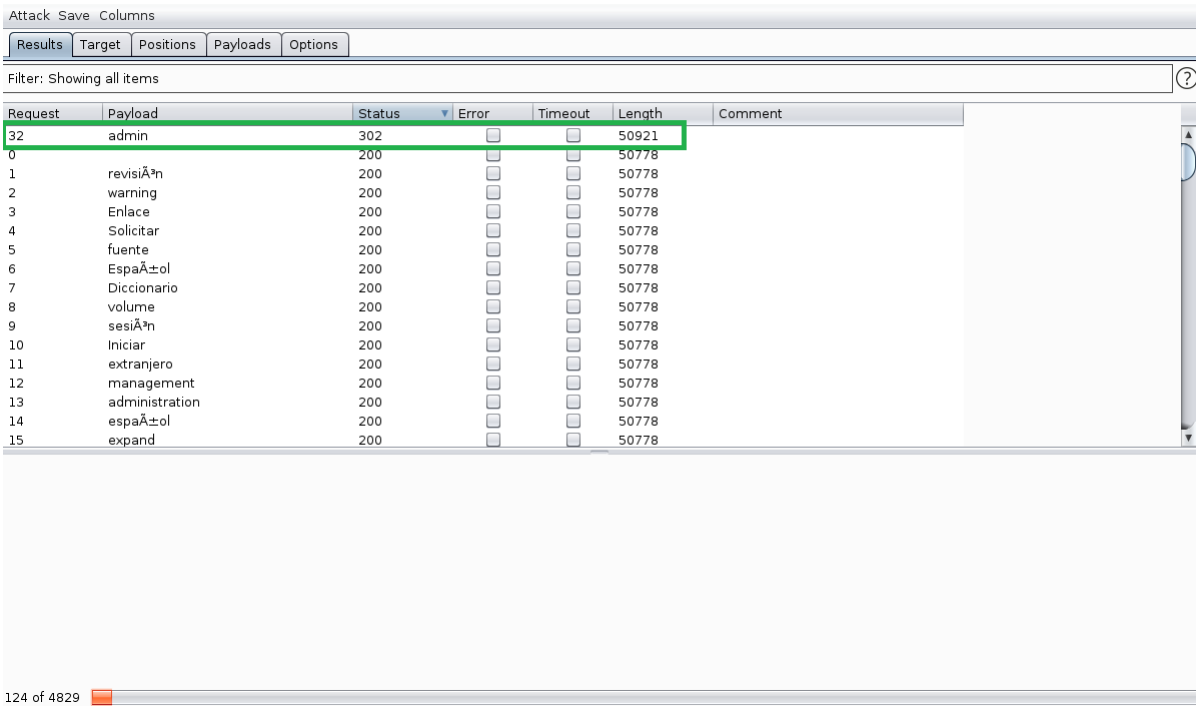

Figura 2.7: Resultado del ataque.

Capítulo 3

Cross Site Scripting (XSS)

Como vimos en las sesiones teóricas, existen dos páginas de entrenamiento online muy buenas para practicar ataques XSS. En esta práctica, resolveremos algunos ejemplos del juego https://xss-quiz.int21h.jp. Este tipo de juegos son útiles para aprender a inyectar XSS de diferentes formas. Además, para los alumnos suele ser divertido y gratificante una vez encuentras la solución.

En la Figura 3.1 encontramos la página de entrenamiento: en el recuadro rojo tenemos el comando a ejecutar, en el azul podemos ver una pista si arrastramos el ratón sobre las letras blancas y el verde es nuestro punto de entrada como jugadores. Este primer ejemplo (referente al Stage 1) es muy sencillo.

Cuando acertemos con la inyección aparecerá algo similar a la Figura 3.2. Tras aceptar el popup, podremos ver el link para el siguiente ejercicio (Figura 3.3). Con esto en cuenta ya podemos tratar de llegar los más lejos posible: el objetivo es la **Stage 6**. En cada una de las secciones siguientes tenéis una explicación de cómo resolver las stages, aunque lo ideal sería probar primero sin las soluciones. Recordad emplear el inspeccionar elemento de vuestro navegador para ver cómo queda el código fuente (si es necesario podéis usar Burp Suite).

3.1. Stage 2. Cerrando etiqueta existente

Si introducimos algo en el cuadro de texto, la salida enviada por el formulario será algo similar a lo que se muestra en la Figura 3.4. Por lo tanto, si cerramos esa etiqueta (con ”>), podemos inyectar lo que queramos a partir de ahí. De esta forma, el payload podría ser:

”><**script**>**alert(document.domain);**</**script**>.

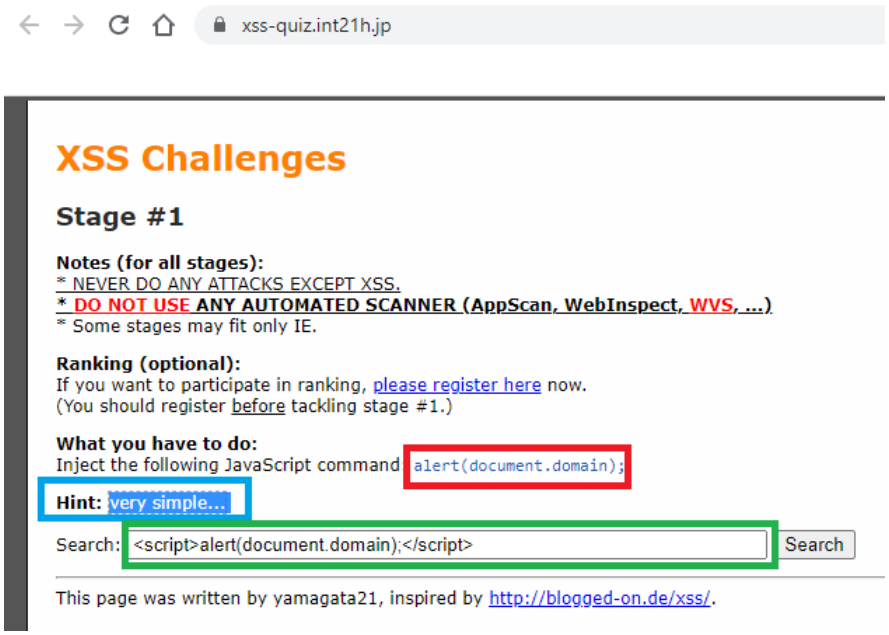

Figura 3.1: Página de entrenamiento.

Figura 3.2: Ejecución del comando correcto.

3.2. Stage 3. Inyectado en desplegables

En este caso, parece que han saneado bien la entrada (como vemos en la Figura 3.5) para buscar un lugar, sin embargo, la elección de países es sospechosa. Usaremos la inspección de elementos (botón derecho sobre el desplegable) para modificar 'Japan' y probarlo como vemos en la Figura 3.6. Una vez tenemos esto, cerramos el inspector y probamos a buscar algún lugar (aleatorio). El resultado será similar a la Figura 3.7.

3.3. Stage 4. Campos ocultos

En este caso, no podemos inyectar ni en el cuadro de texto ni en el desplegable, pero si inspeccionamos la página, vemos que hay algo extraño. Un elemento de tipo oculto como vemos en la Figura 3.8 (tan sospechoso como que el valor del elemento es 'hackme'). Usando el inspector, vamos a introducir el XSS en este campo. Es importante darse cuenta que hay que

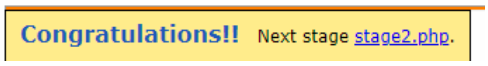

Figura 3.3: Enlace para el siguiente ejercicio.

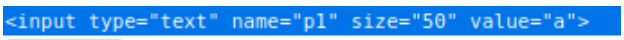

Figura 3.4: Encerrados en en el input.

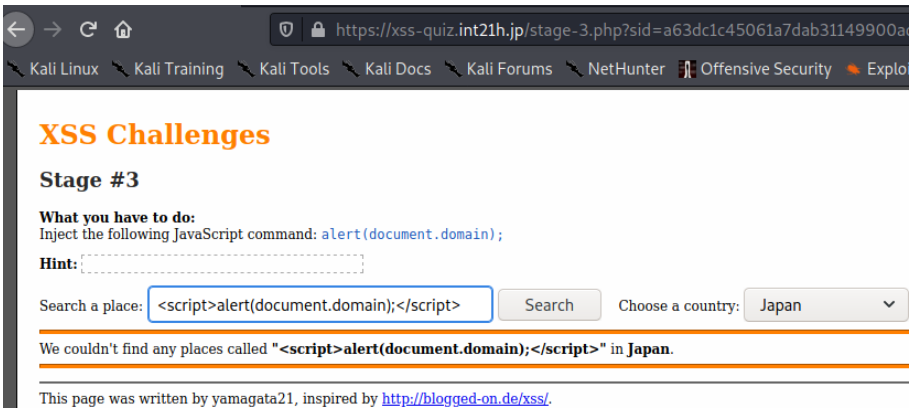

Figura 3.5: Desplegable sospechoso.

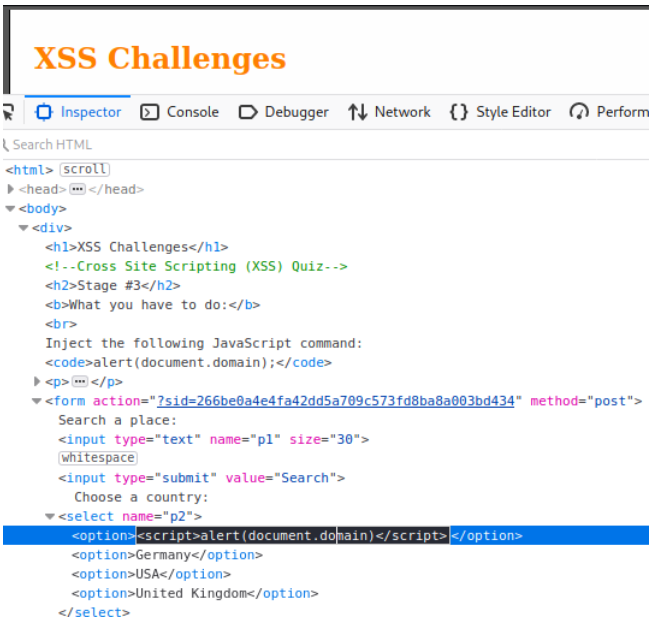

Figura 3.6: Modificando las opciones del desplegable.

escapar del elemento input, como en el Stage 2, sin embargo no sirve emplear directamente las comillas dobles porque no funcionaría con el inspector. En

Figura 3.7: Página modificada.

su lugar, buscaremos el código ASCII de las comillas dobles y emplearemos *&#<numero_ascii>*. Esta etapa es interesante porque se usan dos técnicas juntas. Podemos buscar el código ASCII de las comillas dobles en el siguiente enlace: https://elcodigoascii.com.ar. En la Figura 3.9 podemos ver como se modifica el elemento para introducir el XSS.

```
<input type="hidden" name="p3" value="hackme">
```

Figura 3.8: Elemento oculto.

```
<input type="hidden" name="p3" value="&#34> <script>alert(document.domain);</script>">
```

Figura 3.9: Modificando el elemento oculto para que funcione con el inspector.

3.4. Stage 5. Longitud máxima

La seguridad y los filtros no deben implementarse en la parte del cliente porque pueden modificarse por el atacante. Este ejemplo es una clara muestra de ello. Si tratamos de inyectar en el input, nos damos cuenta de que el tamaño máximo está establecido en 15. La solución es muy sencilla, simplemente modificamos este campo y lo aumentamos. Una vez hecho esto, el ejercicio es similar al del Stage 3 donde escapamos del input. Podemos ver los dos pasos en las Figuras 3.10 y 3.11.

```
▶ <p> ··· </p>
▼ <form action="?sid=bebc5c31e60057b4618792c06293c3177201a117" method="post">
    <hr class="red">
    Search:
    <input type="text" name="p1" maxlength="200" size="30" value="">
    whitespace
    <input type="submit" value="Search">
    <hr class="red">
```

Figura 3.10: Modificando la longitud máxima permitida.

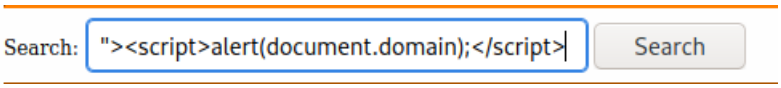

Figura 3.11: Escapamos del input e inyectamos el xss.

3.5. Stage 6. Inyectando con eventos

No siempre es necesaria la etiqueta *<script>* para perpetrar un ataque XSS. En este caso, escapan los caracteres < y >, pero no los demás. Esto nos permite jugar con los eventos de la etiqueta *input*. Primero, escaparemos del campo *value* (similar al Stage 3). Hay que encontrar un evento que podamos accionar, y dentro de ese evento inyectaremos el XSS. Podemos ver un listado de eventos aquí: https://lenguajehtml.com/html/scripting/eventos-html. En este caso, usaremos el evento *onFocus* que se activa cuando el elemento recibe el foco del usuario como puede verse en la Figura 3.12. Una vez que le damos a buscar, tendremos que hacer click otra vez en el input para que reciba el foco y así activar la inyección.

Figura 3.12: Inyectamos con evento en el elemento input.