WordPress DOCS

Pautas básicas para el DESARROLLO DE PLUGINS

1

ÍNDICE

Protección contra CSRF.

Cuando hablamos de CSRF (Cross Site Request Forguery) nos referimos a una vulnerabilidad muy extendida, habitualmente de impacto bajo por sí misma, que consiste en la posibilidad de forzar a un usuario que mantiene una sesión en un sitio web a realizar acciones dentro de ella. Un ejemplo clásico para comprender rápidamente como funciona es el de un link en una web que al ser clickado se realiza una petición a /logout.php y desloguea al usuario. Si nosotros le pasamos ese link al usuario, o le pasamos una web con un <img src=...url/logout.php>, al entrar se deslogueará.

El problema subyace en que no se protegen las acciones que se realizan dentro de la sesión, es decir, no se realiza una comprobación para verificar la legitimidad de quien realiza la acción.

¿Cómo sé que realmente el usuario ha sido quien ha querido hacer click en logout y no ha sido otra persona la que le ha forzado?.

Puede parecer a priori algo trivial esta vulnerabilidad, pero imaginemos que en vez de tratarse de un simple logout, podemos forzar al usuario a que rellene un formulario de configuración de un plugin para backups donde podemos obligarle a que, por ejemplo, ponga en el campo del e-mail donde recibir el backup la dirección del atacante. Eso ya parece más serio, ¿no?.

Volviendo a lo comentado anteriormente, ¿cómo podemos comprobar que la acción es legítima? La respuesta es fácil: añadiendo a la petición una cadena de texto aleatoria que nosotros conocemos y que el atacante no.

De esta forma únicamente si la cadena enviada y la esperada coinciden, se realizará la acción.

Esta "cadena" habitualmente es denominada como token y se generan de forma aleatoria para evitar que un posible atacante pueda llegar a averiguar el patrón y de esta forma hacer un "bypass" de la protección.

Una forma sencilla de hacer esto en PHP es la de generar dicha cadena y almacenarla como una variable de sesión en el momento en el que haga log in el usuario.

Posteriormente deberemos agregar el token en todas las peticiones y realizar la comprobación.

Un ejemplo:

Un ejemplo:

En el PHP donde se realiza el log in:

```
$TOKEN = MD5(BASE64_ENCODE(TIME()));
$TOKEN = $TOKEN.RAND();
$_SESSION['TOKEN'] = MD5($TOKEN.$TIME().$TOKEN);
```

Después, para proteger un formulario, por ejemplo, se podría enviar esta cadena como
campo hidden:

```
ECHO '<INPUT TYPE="HIDDEN" NAME="TOKEN" VALUE=".$_SESSION['TOKEN']."">';
```

Y por último sólo quedaría comprobar que la petición post contiene el token necesario:

```
IF(... AND $_POST['TOKEN'] === $_SESSION['TOKEN']) {
```

Nótese que se utilizan 3 = y no 2.

Por suerte para los desarrolladores WordPress posee en su API una serie de funciones destinadas a cumplir esta función de generación de tokens. Estas funciones permiten crear tokens, o "nonces", de forma sencilla y posteriormente verificar su validez.

Para la generación de los nonces, existe la función wp_createnonce() que recibe como parámetro una cadena que identifica la acción que se desea de proteger.

```
$NONCE = WP_CREATE_NONCE( 'LOGIN' );
```

De esta forma la variable $nonce contendría nuestra cadena aleatoria y tan sólo quedaría añadirla a la petición que deseamos. Existen otras funciones que permiten añadir el nonce directamente a la petición deseada (si es GET, wp_nonce_url() y si es POST wp_nonce_field()).

Para comprobar si el "nonce" es válido podemos usar la función wp_nonce_verify(), que comprobará si el nonce es el corrrecto:

```
$NONCE = $_RESQUEST['NONCE'];
IF (! WP_NONCE_VERIFY( $NONCE, 'LOGIN'))
{ EXIT("ERROR, NONCE NOT VALID");
} ELSE { //DO STUFF }
```

Como protección extra podemos recurrir además a la comprobación del "referer" de la petición HTTP, de tal forma que podemos comprobar si la petición procede de un lugar esperado (del propio WordPress) o no. Si la petición procede de una web externa, es obviamente una petición que no deseamos tramitar. Para realizar dicha comprobaación podemos usar wp_referer_field().

Protección XSS

Cuando se permite la introducción de datos por parte de los usuarios y posteriormente se muestra tal cual, podemos estar ante un caso de XSS o Cross Site Scripting. Esta vulnerabilidad radica en la posibilidad de inyectar scripts en el HTML de la web, de tal forma que puede controlar el navegador de aquel que visualice ese HTML. La forma más habitual de explotarlo es ejecutando JavaScript, y el objetivo es variado (robo de sesiones, distribución de malware, etc.).

Para poder prevenir este tipo de vulnerabilidades en nuestro código deberemos de filtrar todas las variables que cuyo contenido es asignado en base a los valores que el usuario haya puesto. El no limpiar o escapar ciertos caracteres puede permitir al atacante introducir su código JavaScript. Por suerte la API de WordPress posee una serie de funciones que van a permitirnos escapar aquellos caracteres que pueden suponer complicaciones.

Estas funciones poseen pequeños matices que las hacen más efectivas para diferentes escenarios. Algunas de éstas son:

- esc_js: limpia javascript.
- esc_url: limpia una URL, eliminando caracteres inválidos o peligrosos y codificando el resto.
- esc_html: codifica los caracteres < > & " .

Siempre hay que filtrar los datos antes de trabajar con ellos, tanto a nivel de "inputs" (valores que se asignan con lo que el usuario envía) como de "outputs" (valores que son asignados a partir de información almacenada en la base de datos).

Protección contra inyecciones SQL

Al igual que ocurría con los XSS, en el caso de las inyecciones SQL (SQLi) el problema viene por no realizar un correcto filtrado de los datos que se van a introducir en la base de datos, de tal forma que un atacante puede modificar la petición SQL original y controlarla para que realice otra arbitraria, tomando control total de la base de datos.

En primer lugar podemos realizar una limpieza de las variables usando la función esc_sql que viene en el propio wordpress; pero debemos de tener en cuenta que es preferible realizar esta tarea usando $wpdb->prepare(), ya que además de filtrar adecuadamente la petición corrige otros posibles errores en estas.

En el caso de estar esperando un dígito como valor podemos forzar la conversión a un int para evitar que se introduzca texto usando la función intval($var) .

Consideraciones menores en el manejo de datos

Cuando los datos que estamos esperando recibir los conocemos y son pocos, podemos establecer una politica de "white list" y comprobar si el valor enviado por el usuario es uno de los esperados, y si lo es, actuar en consecuencia. Igualmente es una buena práctica comprobar si el dato que estamos esperando es del tipo que queremos. Por ejemplo, si estamos esperando recibir un número podemos usar is_numeric(), o si se trata de una dirección de correo electrónico is_email().

Más genérico, y orientado a evitar la introducción de caracteres no alfanuméricos es la comprobación mediante ctype_alnum().

Otra posibilidad a la hora de tratar los datos, diferente a la política de comprobación, es permitir insertar cualquier cadena y tratar de limpiarla usando expresiones regulares a través de funciones tipo preg_replace. De esta forma el resultado final no contendrá caracteres peligrosos.

WordPressA DOCS

www.wordpressa/quantika14.com

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

Tlf: +34 605 938 908