Seguridad Informática: Capas de Seguridad Inteligentes en PHP – Saneamiento del array superglobal $_POST

La siguiente publicación, corresponde a los resultados obtenidos hasta el momento, sobre una investigación que me encuentro realizando actualmente, cuyo objetivo es hallar una forma inequívoca de sanear, filtrar y limpiar datos enviados por HTTP POST, mediante una capa de seguridad de activación automática.

En varias ediciones de la revista Hackers & Developers Magazine, he dado sugerencias alternativas para el saneamiento de los datos recibidos desde formularios. En http://library.originalhacker.org se puede acceder a cada uno de los artículos sobre el tema y descargarlos en formato PDF.

Pero lo que hoy quiero mostrarles, no es en referencia a qué datos deben filtrarse, qué riesgos corremos a causa de cuáles vulnerabilidades conocidas ni qué herramientas existen para prevenir estos problemas.

Hoy, quiero hablarles sobre **una investigación que vengo llevando a cabo en los últimos meses**, mediante la cual, **valiéndome de una característica de PHP** que personalmente no logra convencerme **y de las raíces más básicas de la inteligencia artificial**, es posible **crear capas de seguridad** en nuestras aplicaciones, **que se activen de forma automática** y **que sean capaces de reconocer el tipo de información** que el programador espera y sobre ese análisis, procedan a "limpiar" el *array superglobal* $_POST de forma tal que al ser accedido por el programador, ya haya atravesado toda acción de saneamiento.

Mutabilidad del array $_POST

En el párrafo anterior comentaba acerca de una característica de PHP que aún no lograba convencerme. Sin embargo, al final de este documento, probablemente termine reconociendo lo contrario -o no-. Se trata de la **capacidad mutable del *array superglobal* $_POST**.

Uno podría esperar que se tratase de una variable de solo lectura, pero no lo es. De hecho, **$_POST no almacena los datos en crudo enviados mediante HTTP POST**, sino que lo hace habiéndolos tratado previamente. **Un claro ejemplo de esto, es lo que antiguamente sucedía con los datos cuando las famosas comillas mágicas se encontraban activadas.** De forma automática, los datos enviados mediante HTTP POST llegaban al *array superglobal* $_POST con las comillas dobles, simples, barras invertidas y valores nulos ya escapados con su barra invertida correspondiente.

Retomando el caso de mutabilidad de este *array*, la misma nos permite, por ejemplo, agregar nuevas claves al *array*, como se muestra a continuación:

```
$_POST['esto_no_era_un_campo_del_form'] = 'foo';
```

Y como era de esperarse, también nos permite modificar un valor existente:

```
$_POST['campo_del_form'] = htmlentities($_POST['campo_del_form']);
```

Esta última característica, es la que más nos interesa. Gracias a esta falta de inmutabilidad del *array*, podremos hacer que esté disponible con los datos ya saneados.

Automatización

Para que el proceso de saneamiento que se lleve a cabo se active de forma automática, éste debería ser invocado desde un archivo común a toda la aplicación, que siempre se encuentre disponible y que tenga relación directa con esta capa. Por ejemplo, una clase `Template` siempre podría estar disponible y sin embargo, no tendría ninguna relación con una capa de seguridad.

Después de mucho pensar, concluí en que debía ser la propia capa de seguridad quien se active a sí misma. De esta forma, al momento de ser importada (por ejemplo, desde un `settings`), ya estaría actuando de forma natural.

No obstante, también noté la necesidad de poder decidir la desactivación de esta capa si fuese requerido sin tener que estar comentando o eliminando la línea de importanción. Así fue que lo más coherente, me pareció que sería definir una constante *booleana* en un archivo `settings` y dependiendo de su valor, se activara o no la capa.

Disponibilidad de los datos originales

La inteligencia artificial y la lógica de negocio de una aplicación, deben siempre estar al servicio de las personas y jamás debería darse el caso contrario (aunque lamentablemente, se da en más del 80% de las aplicaciones, pero eso, ya no es tema de debate en este artículo).

Después de probar múltiples alternativas, analizarlas una a una y descubrir que ninguna me terminaba de convencer, decidí recorrer la documentación de PHP de punta a punta. Ya había descartado el uso de la variable `$HTTP_RAW_POST_DATA` puesto que dependía del valor de la directiva `always_populate_raw_post_data` del archivo `php.ini` y realmente, me resultaba poco lógico e inviable, tener que modificar esto para poder disponer de los datos HTTP POST puros. Leyendo sobre esta directiva, fue que la solución llegó a mis manos: **`php://input`**, una envoltura de **flujo de solo lectura**.

Sin embargo, cabe destacar que `php://input` cuenta con algunas restricciones interesantes, como por ejemplo, su no disponibilidad cuando el tipo de codificación del formulario es declarado como `multipart/form-data`.

`php://input NO FUNCIONA con enctype="multipart/form-data"`

Por favor, recordemos que el tipo de contenido `multipart/form-data`, es el que nos permite en un formulario, enviar grandes cantidades de datos sobre todo binarios y con codificación diferente a ASCII.
Para mayor información sobre este tema, recomiendo dirigirse a
http://www.w3.org/TR/html401/interact/forms.html#didx-multipartform-data y
http://www.ietf.org/rfc/rfc2388.txt.

A pesar de todo lo anterior, esta solución resulta la más acertada posible. Cuando `php://input` no se encuentre disponible, habría que pensar entonces, en recurrir como última alternativa, a la variable `$HTTP_RAW_POST_DAT` que podría también no estar disponible si la directiva `always_populate_raw_post_data` del archivo `php.ini` fuese `False`.

> *Para ser honesta, en PHP no existe una forma inequívoca y exacta de poder recuperar los datos enviados a través de HTTP POST sin previo tratamiento.*

Si se quisiera trabajar con `php://input`, debería hacerse mediante `fopen` y `fgets` como se muestra en el siguiente ejemplo:

```
$fp = fopen('php://input', 'r');
$data = fgets($fp);
fclose($fp);

# Donde la variable $data, podría retornar algo como:
foo=Hola+mundo%21&bar=0.65

# Equivalente a haber enviado por POST:
foo = Hola mundo!
bar = 0.65
```

Inteligencia Artificial

Para saber cómo se debería filtrar cada uno de los campos y qué filtros aplicar, podemos recurrir a las raíces más básicas de la inteligencia artificial: los diccionarios de nombres. De esta forma, podemos deducir que un campo que solicite un e-mail al usuario, podría tener como parte del nombre, la cadena "`mail`". Así, campos llamados "email", "mail_address" o "repeat_email" coincidirían con la cadena "mail". La misma lógica, podría aplicarse a otros tipos de campo:

```
IP                      ip
URL                     url, uri, site, web
Nombre de usuario       uname, username, user, usuario
Contraseña              pass (coincidiría además con passwd y password), pwd, clave
etc.
```

Claro que esta lógica no debe ser demasiado minuciosa. Es decir, la mayoría de los campos serán campos de texto (*strings*) que no van a necesitar de comprobaciones de formatos especiales.

De esta forma, creando diccionarios y aplicando a cada caso los filtros correspondientes, se podría abarcar un gran número de posibilidades que podrían sanear de forma predeterminada a decenas de miles de aplicaciones.

Implementación de la IA

Ya en muchos documentos anteriores me dediqué a hablar de los filtros que se deben aplicar en cada caso y en cada tipo de campo, así que aquí solo me concentraré en poner en código todo lo que expliqué anteriormente referido a la IA.

Dado un diccionario como el siguiente:

```
$diccionarios = array(
    'e-mail' => array('mail'),
    'contraseña' => array('pass', 'clave')
);
```

Y suponiendo un formulario con los siguientes campos y valores:

```
$_POST = array(
    'email_address' => 'foo@bar',
    'passwd' => '123456',
    'passwd2' => '123456'
);
```

El siguiente algoritmo podría utilizarse para detectar a qué tipo de dato se correspondería cada uno de los campos:

```
foreach($_POST as $key=>$value) {
    foreach($diccionarios as $tipo=>$diccionario) {
        foreach($diccionario as $entrada) {
            if(strpos($key, $entrada) !== False) {
                print "El campo {$key} es de tipo {$tipo}" . chr(10);
            }
        }
    }
}
```

Obteniendo un resultado como el siguiente:

```
El campo email_address es de tipo e-mail
El campo passwd es de tipo contraseña
El campo passwd2 es de tipo contraseña
```

Luego, basados en el tipo de campo, solo es cuestión de aplicar los filtros que correspondan en cada caso.

Más información

Para conocer más sobre filtros en PHP, recomiendo leer las siguientes páginas del manual:

- Función filter_input:
  http://us3.php.net/manual/es/function.filter-input.php
- Filtros disponibles:
  http://us3.php.net/manual/es/filter.filters.php
- Función strip_tags:
  http://us3.php.net/manual/es/function.strip-tags.php
- Función htmlentities:
  http://us3.php.net/manual/es/function.htmlentities.php
- Función htmlspecialchars:
  http://php.net/manual/es/function.htmlspecialchars.php