WHK Security

Programación general => Desarrollo WEB => PHP => Mensaje iniciado por: WHK en enero 16, 2013, 10:37:31 am

---

Título: **Inyección NoSQL en MongoDB + PHP**
Publicado por: **WHK** en **enero 16, 2013, 10:37:31 am**

Los tiempos cambian y MySQL cada dia es menos utilizado. Hoy por hoy todos los que están dejando MySQL están migrando a MongoDB el cual pareciera ser el futuro de las bases de datos de código abierto, lo pueden comprobar mirando la cantidad infinita de foros y portales que lo enseñan y recomiendan a nivel básico y profesional (incluso dicen que facebook lo utiliza por su rapidez y eficiencia).

Por este motivo también me estoy migrando a MongoDB pero no puedo comenzar a crear aplicaciones estables y seguras si no se que tipo de precauciones debo tener.

Conversando con 3l3cTr0n1k_0 (http://whk.drawcoders.net /index.php?action=profile;u=106) le pregunté.. "existirán inyecciones sql en nosql con mongodb?" e inmediatamente abrió su explorador y buscó en google sobre inyecciones de este tipo y nos encontramos con la sorpresa de que **si existen** :D

http://www.idontplaydarts.com/2010/07/mongodb-is-vulnerable-to-sql-injection-in-php-at-least/

Tal como lo demuestra este enlace mongoDB ya no procesa sentencias escritas como sql sino que trabaja directamente con objetos por lo tanto cuando le ingresas un string 'or1=1 te lo escribe muy bien, pero como trabajamos con objetos debemos tener cuidado de que cosa le estamos pasando al motor debido a que php tiene la ventaja de poder recibir variables en formas de arrays como por ejemplo test.php?algo[abc]=abc&algo[def]=def y para mongoDB eso le significa una inserción de datos arbitraria sobre columnas inexistentes las cuales serán creadas nuevas.

Pero vamos al grano... les haré la traducción de lo explicado en este sitio web que les mensioné anteriormente:

MongoDB es vulnerable a la inyección de SQL por lo menps en PHP

Es un error común pensar que como MongoDB no utiliza SQL no es vulnerable a ataques de inyección SQL. PHP utiliza objetos en lugar de un Servidor SQL para pasar consultas al servidor MongoDB, como por ejemplo la secuencia de comandos siguiente se selecciona un elemento en forma MongoDB el nombre de usuario es igual a '**bob**' y la contraseña es igual a '**password**'.

Código: php [Seleccionar]

```
$collection->find(array(
```

```
  "username"    => $_GET['username'],
  "passwd"    => $_GET['passwd']
));
```

Esto es equivalente a la sintaxis SQL:

Código: php [Seleccionar]

```
mysql_query("
  SELECT * FROM collection
  WHERE username=" . $_GET['username'] . ",
  AND passwd=" . $_GET['passwd']
)
```

En un ataque de inyección normal de SQL que puede sustituir a uno de los dos parámetros de entrada con una cadena de tal manera que la consulta SQL siempre devuelve verdadero. Por ejemplo:

Código: [Seleccionar]

```
login.php?username=admin&passwd=" OR 1 --
```

Eso no funcionará con MongoDB, pero si podemos pasar un objeto al conductor MongoDB PHP que podría alterar la consulta de una manera similar. Por suerte PHP nos proporciona una manera de pasar objetos como parámetros GET o POST:

Código: [Seleccionar]

```
login.php?username=admin&passwd[$ne]=1
```

Esto crea la consulta MongoDB:

Código: php [Seleccionar]

```
$collection->find(array(
  "username"    => "admin",
  "passwd"    => array("$ne" => 1)
));
```

El cual es el equivalente a la siguiente instrucción SQL que, a menos que la contraseña es "1" siempre devolverá true:

Código: php [Seleccionar]

```
mysql_query("
  SELECT * FROM collection
  WHERE username="admin",
  AND passwd!=1
");
```

La solución es asegurarse de que sus variables están correctamente escrito antes de que pasen al conductor MongoDB. El siguiente código no es vulnerable a la inyección de MongoDB:

Código: php [Seleccionar]

```
$collection->find(array(
    "username"    => (string)$_GET['username'],
    "passwd"    => (string)$_GET['passwd']
));
```

Bueno, despues de haber leido este fantastico documento del blog de Phil vamos a profundizar un poco mas sobre "**porqué un cast string evita la inyección**".

http://php.net/manual/es/language.types.type-juggling.php#language.types.typecasting

A traves del sitio oficial de PHP podemos los diferentes tipos de datos, pero ¿para que sirven?

Código: php [Seleccionar]

```
$a = '000123456abc';
echo (int)$a; // 123456
```

Un cast nos permite decirle a php que tipo de datos exactamente es una variable y debe comportarse de esa forma. La ventaja de php spobre ASP es que no te arroja un error cada ves que quieres procesar un integer ni necesitas hacer verificaciones anteriores, solo le dices que será una variable de tipo integer y automaticamente tomará el valor numérico y el resto lo desechará. Si vamos a procesar decimales entonces utilizamos (float).

Con esta pequeña introducción ya masomenos podemos entender para que nos sirve el cast string, o sea el (string)$algo.

Código: php [Seleccionar]

```
$a = array('a' => 'b');
echo (string)$a; // Array
```

Si $a es un array entonces al castear por string nos devolverá un texto que dice simplemente "Array", de hecho este es un tipo de fallas de seguridad al momento de programar sin utilizar cast porque a veces utilizamos funciones que solo aceptan un string pero por cosas de la vida le pasamos un array y php mostró un warning:

Código: php [Seleccionar]

```
$a = $_GET['x'];
echo htmlspecialchars($a, ENT_QUOTES);
```

Si ejecutamos index.php?x=hola no habrá ningún problema, pero si le pasamos un array index.php?x[]=x mostrará un arning ya que la función htmlspecialchars solo acepta strings y no arrays.

En conslusión... se castea por String para evitar que se pasen arrays al objeto de MongoDB y evitar inyecciones que pueda causar que un atacante evada consultas de seguridad accediendo a secciones no autorizadas.

Saludos.