【译】WordPress WPDB SQL注入攻击

angel010 / 2017-11-03 15:27:00 / 浏览数 2599 技术文章 技术文章 顶(0) 踩(0)

WordPress 4.8.3中修复了一个重要的SQL注入漏洞。漏洞是今年9月20日由Hacker-One报告的。本文主要讲了漏洞的技术细节和解决方法。

升级到最新版本

网站管理员应该升级WordPress到4.8.3版本并更新重写$wpdb的所有插件，就可以预防此类问题。为客户机升级wp-db.php，同时可能需要修改一些防火墙规则，比如拦截
 和其他`sprintf()`值。

插件开发者应该？

一般来说 `->prepare()`检查过滤`$query`参数，不会将用户输入传递到查询端，如：

```
$where = $wpdb->prepare(" WHERE foo = %s", $_GET['data']);
$query = $wpdb->prepare("SELECT * FROM something $where LIMIT %d, %d", 1, 2);

$where = "WHERE foo = '" . esc_sql($_GET['data']) . "'";
$query = $wpdb->prepare("SELECT * FROM something $where LIMIT %d, %d", 1, 2);
```

以上两种方法从概念是讲都是不安全的，要分别构建查询和参数，然后一次性查询

安全的查询方法为：

```
$where = "WHERE foo = %s";
$args = [$_GET['data']];
$args[] = 1;
$args[] = 2;
$query = $wpdb->prepare("SELECT * FROM something $where LIMIT %d, %d", $args);
```

漏洞

`WPDB::prepare`源码（4.8.2之前版本）：

```
public function prepare( $query, $args ) {
   if ( is_null( $query ) )
       return;
   // This is not meant to be foolproof -- but it will catch obviously incorrect usage.
   if ( strpos( $query, '%' ) === false ) {
       _doing_it_wrong( 'wpdb::prepare', sprintf( __( 'The query argument of %s must have a placeholder.' ), 'wpdb::prepare()'
   }
   $args = func_get_args();
   array_shift( $args );
   // If args were passed as an array (as in vsprintf), move them up
   if ( isset( $args[0] ) && is_array($args[0]) )
       $args = $args[0];
   $query = str_replace( "'%s'", '%s', $query ); // in case someone mistakenly already singlequoted it
   $query = str_replace( '"%s"', '%s', $query ); // doublequote unquoting
   $query = preg_replace( '|(?<!%)%f|' , '%F', $query ); // Force floats to be locale unaware
   $query = preg_replace( '|(?<!%)%s|', "'%s'", $query ); // quote the strings, avoiding escaped strings like %%s
   array_walk( $args, array( $this, 'escape_by_ref' ) );
   return @vsprintf( $query, $args );
}
```

1、用`vsprintf`（与`sprintf`基本等价）的值来替换占位符；

2、用`str_replace`来适当地引用占位符；

3、如果传递了一个参数，而这个参数是数组的话，用数组的值来替换参数。

这意味着调用`$wpdb->prepare($sql, [1, 2])`与调用`$wpdb->prepare($sql, 1, 2)`是等价的。

最初报告的漏洞依赖与下面的服务端代码：

```
$items = implode(", ", array_map([$wpdb, '_real_escape'], $_GET['items']));
$sql = "SELECT * FROM foo WHERE bar IN ($items) AND baz = %s";

$query = $wpdb->prepare($sql, $_GET['baz']);
```

漏洞利用`vsprintf`的特征来允许绝对引用参数，例子如下：

```
vsprintf('%s, %d, %s', ["a", 1, "b"]); // "a, 1, b"
vsprintf('%s, %d, %1$s', ["a", 2, "b"]); // "a, 2, a"
```

注意`%n$s`不会读下一个参数，但是会读第n个位置的参数。可以根据这个特性在原始查询中进行注入。假设传递下面的信息到请求中：

```
$_GET['items'] = ['%1$s'];
$_GET['baz'] = "test";
```

查询会变成

```
SELECT * FROM foo WHERE bar IN ('test') AND baz = 'test';
```

我们成功地改变了查询的本意。

最初的漏洞报告中还有一个关键的信息是可以把这个变成成熟的SQL注入。Sprintf也会接受其他类型的参数，%c与chr()含义相同，可以把小叔变成字符，所以攻击者可以：

```
$_GET['items'] = ['%1$c) OR 1 = 1 /*'];
$_GET['baz'] = 39;
```

ASCII表中39代表’（单引号），所以查询就变成了这样：

```
SELECT * FROM foo WHERE bar IN ('') OR 1 = 1 /*' AND baz = 'test';
```

注入就完成了。

这个过程看似很复杂，需要提前准备好输入的参数等，实际上该漏洞也存在于核心文件/wp-includes/meta.php 中：

```
if ( $delete_all ) {
 $value_clause = '';
 if ( '' !== $meta_value && null !== $meta_value && false !== $meta_value ) {
   $value_clause = $wpdb->prepare( " AND meta_value = %s", $meta_value );
 }
 $object_ids = $wpdb->get_col( $wpdb->prepare( "SELECT $type_column FROM $table WHERE meta_key = %s $value_clause", $meta_key
}
```

最早的补丁

WordPress4.8.2发布时，就包含上述问题的一个补丁。补丁整个包含在`WPDB::prepare()`中，补丁只加了1行代码：

```
$query = preg_replace( '/%(?:%|$|([^dsF]))/', '%%\\1', $query );
```

这1行代码做了2件事情。

1. 是移除了除`%d`，`%s`，`%F`之外的`sprintf`令牌，因为漏洞是依赖`%c`的，因此使漏洞无效。
2. 是移除了位置替换的能力，即`%1$s`这样的参数就无效了。

这引起了开发人员的不满，因为WordPress在官方文档中说只能使用`%d`，`%s`，`%F`。即使官方文档是这么写的，上百万的第三方查询代码都使用了前面的语法规则。

WordPress的回应是“won’t fix, sorry”，并以安全为由拒绝提供更多细节。

最初补丁的第一个问题

漏洞是传递用户输入到prepare的服务端。最初漏洞的POC是这样的，安全查询代码如下：

```
$db->prepare("SELECT * FROM foo WHERE name= '%4s' AND user_id = %d", $_GET['name'], get_current_user_id());
```

4.8.2中的变化是`%4s`会被重写成`%%4s`，也就是说`%d`会反弹到`$_GET['name']`，给了攻击者用户`id`的机会。这可以被用来进行权限提升攻击。

Wordpress的回应是：“thank you, we don’t support that”。

全面攻击

然后作者设计了一个不同的POC，利用另一个重要的事实来证明该漏洞不是%1$s，而是传递用户输入到prepare查询端。Meta.php文件代码如下：

```
if ( $delete_all ) {
 $value_clause = '';
 if ( '' !== $meta_value && null !== $meta_value && false !== $meta_value ) {
   $value_clause = $wpdb->prepare( " AND meta_value = %s", $meta_value );
 }
 $object_ids = $wpdb->get_col( $wpdb->prepare( "SELECT $type_column FROM $table WHERE meta_key = %s $value_clause", $meta_key
}
```

输入：

```
$meta_value = ' %s ';
$meta_key = ['dump', ' OR 1=1 /*'];
```

产生了下面的查询：

```
SELECT type FROM table WHERE meta_key = 'dump' AND meta_value = '' OR 1=1 /*'
```

成功注入了核心文件，$meta_value 和 $meta_key都来自于用户的输入。会产生下面的赋值子句：

```
AND meta_value = ' %s '
```

未引用的%s通过prepare被引用的%代替，第二次调用->prepare()把clause变成AND meta_value = ' '%s' '，就可以注入了。

作者强调该漏洞不能在WPDB::prepare()修复，但是是meta.php中的问题。可以通过预防double prepare calls缓解该漏洞。但是不能修复原始漏洞。

简单补丁

简单的补丁不是传递用户输入的$query参数到meta.php中的WPDB::prepare()。传递用户输入到$query是错误的。

缓解补丁

下一步是在预查询中引用占位符，然后在执行查询前恢复占位符，这个补丁已经有了。基本上，补丁会修改WPDB::prepare()把随机字数穿用%占位符代替，比如：

```
$query = str_replace('%', "{$this->placeholder_escape}", $query );
```

然后，在WPDB::_do_query()去除占位符来恢复最初的用户的用户输入。

我仍然认为传递用户输入到prepare的查询端是存在潜在危险的而且是不安全的。即使你解决了已知的安全漏洞，double-preparing字符串是及其危险的，因为prepar

正确的补丁

正确的补丁应该是抛弃整个prepare机制。像正常的查询那样，返回一个statement或query的对象，或者直接执行查询。这种方式可以预防double prepare字符串的情况。值得一提的是这将会是WP的主要变化。其他平台已经有成功的先例了，比如PHPBB经历了同样的事情，从大规模的SQL注入漏洞到几乎没有SQL注prepared statements。这些变化并不能防止被误用，但是会让误用变得更难。

https://blog.ircmaxell.com/2017/10/disclosure-wordpress-wpdb-sql-injection-technical.html

上一篇：测试特 下一篇：【解读】NTT Security ...

1. 0 条回复
   - 动动手指，沙发就是你的了！

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板