CVE-2016-5734 phpmyadmin后台代码执行漏洞复现

邑安科技 邑安全 今天

0x01 漏洞简介

Phpmyadmin是一个以php为基础，以Web-Base方式架构在网站主机上的MySQL的数据库管理工具，让管理者可以使用Web接口管理MySQL数据库。借由次Web接口可以成为一个简单方式输入SQL语法的较佳途径。其优势就是可以通过Web界面来远程管理方便建立、修改、删除数据库及资料表

0x02 漏洞概述（就是漏洞出自于哪里）

Php中的 preg_replace 函数 该函数是执行一个正则表达式并实现字符串的搜索与替换。

preg_replace (mixed $pattern , mixed $replacement , mixed $subject [, int $limit = -1 [, int &$count]])

搜索 subject 中匹配 pattern 的部分， 以 replacement 进行替换。

参数说明：

```
$pattern: 要搜索的模式，可以是字符串或一个字符串数组。反斜杠定界符尽量不要使用，而是使用 # 或者 ~
$replacement: 用于替换的字符串或字符串数组。
$subject: 要搜索替换的目标字符串或字符串数组。
$limit: 可选，对于每个模式用于每个 subject 字符串的最大可替换次数。默认是-1（无限制）。
$count: 可选，为替换执行的次数。
```

该函数的返回值：当$subject为一数组的情况下返回一个数组，其余情况返回字符串。
匹配成功则将替换后的subject被返回，不成功则返回没有改变的subject，发生语法错误等，返回NULL。

```php
<?php
echo "<p>CVE-2016-5734</p>";
@$demo = $_GET['id'];
$pattern = "~test~";
$replacement= "world";
$subject = "hello test";

var_dump(preg_replace($pattern,$replacement,$subject));
echo "<p></p>".$replacement;

?>
```


CVE-2016-5734

string(11) "hello world"

world

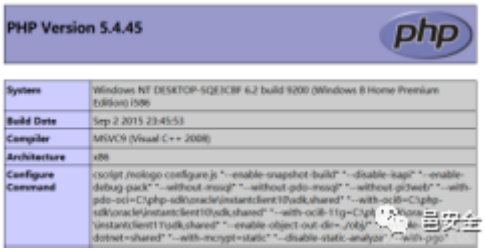
正则表达式修正符：
因为`$pattern`中指定的是要搜索的模式字符串，一般使用的是正则表达式，正则表达式中存在修正符，像/i 就是指定取消大小写敏感，等。具体可参考：
但是其中一个修正符 “/e”;在替换字符串中对逆向引用作正常的替换，将其作为 PHP 代码求值，并用其结果来替换所搜索的字符串。

php代码执行：

```php
<?php
echo "<p>CVE-2016-5734</p>";
@$demo = $_GET['id'];
$pattern = "/(test)/e";
$replacement= "phpinfo();";
$subject = "hello test/e";

var_dump(preg_replace($pattern,$replacement,$subject));
//echo "<p></p>".$replacement;

?>
```

可以看到，使用/e修正符的同时在 Subject 中成功匹配，replacement部分被当作php 代码执行。
这个函数是CTF代码审计中的常客；

0x03 影响版本

Phpmyadmin -> 4.0.10.16之前的4.0.x版本
4.4.15.7 之前的 4.4.x版本

4.6.3之前的 4.6.x版本
Php版本：4.3.0 ~5.4.6
Php 5.0 版本以上的将 preg_replace 的 /e修饰符给废弃掉了

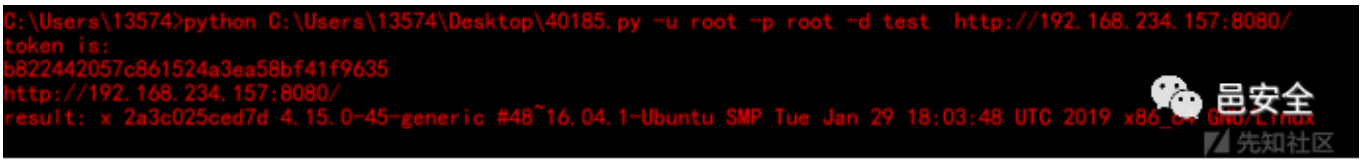
0x04 环境搭建

虚拟机环境 ：
IP：192.168.234.157
使用的是Docker + Docker-compose 开源项目 vulhub
这里使用的是 phpmyadmin 的 4.4.15.6版本

端口 8080

0x05 漏洞复现

1.使用前提是登陆的情况下
2.复现
漏洞poc ： exploit-DB
使用exploit 上面提供的 poc 进行操作
命令 python 40185.py -u root -p root -d test http://192.168.234.157:8080
其中可以使用 -c 指定PHP 代码执行（这里未指定使用代码中默认的system('uname -a')）
-d 指定数据库名
-t 指定用户所创建的表名（这里未指定使用代码中默认的）
结果显示:result的那一行



0x06 漏洞触发点

查询资料：
首先找到preg_replace()函数的调用位置：
发现是在 /libraries/TableSearch.class.php 文件中，

```php
function _getRegexReplaceRows($columnIndex, $find, $replaceWith, $charSet) //这里find作为参数 回溯该类的方法
{
    $column = $this->_columnNames[$columnIndex];
    $sql_query = "SELECT "
        . PMA_Util::backquote($column) . ","
        . " 1," // to add an extra column that will have replaced value
        . " COUNT(*)"
        . " FROM " . PMA_Util::backquote($this->_db)
        . "." . PMA_Util::backquote($this->_table)
        . " WHERE " . PMA_Util::backquote($column)
        . " RLIKE '" . PMA_Util::sqlAddSlashes($find) . "' COLLATE "
        . $charSet . "_bin"; // here we
        // change the collation of the 2nd operand to a case sensitive
        // binary collation to make sure that the comparison is case sensitive
    $sql_query .= " GROUP BY " . PMA_Util::backquote($column)
        . " ORDER BY " . PMA_Util::backquote($column) . " ASC";

    $result = $GLOBALS['dbi']->fetchResult($sql_query, 0);

    if (is_array($result)) {
        foreach ($result as $index=>$row) {
            $result[$index][1] = preg_replace(
                "/" . $find . "/",    //回溯find函数
                $replaceWith,
                $row[0]
            );
        }
    }
    return $result;
}
```


可以看到 _getRegplaceRows()函数中 将find参数传入，并且将find参数作为preg_replace()函数的第一个参数使用；我们既然要构造payload 就需要将这三个参数 find 、replaceWith、row[0]全部溯源查看；

首先对_getRegplaceRows函数进行溯源：

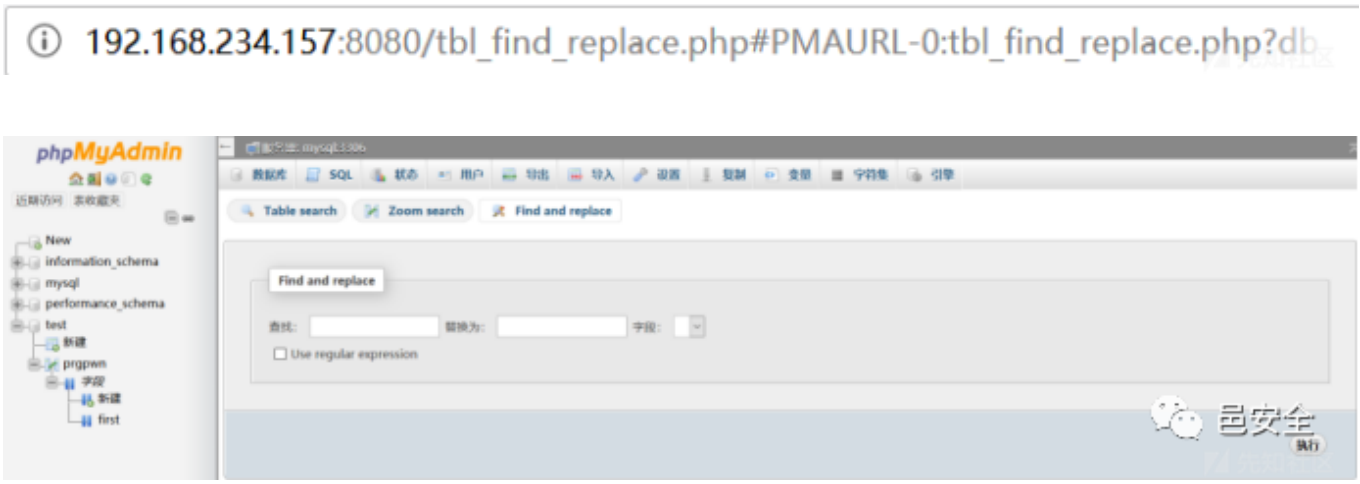
可以看出在同一文件下_getRegplaceRows 被 getReplacePrevies 这个类的方法所调用，并且find参数与replacement参数都是经过该方法所传递的，在对这个函数进行溯源；

```php
if (isset($_POST['find'])) {
    $preview = $table_search->getReplacePreview(      //发现在这个地方
        $_POST['columnIndex'],
        $_POST['find'],
        $_POST['replaceWith'],
        $_POST['useRegex'],
        $connectionCharSet
    );
    $response->addJSON('preview', $preview);
    exit;
}

$header  = $response->getHeader();
$scripts = $header->getScripts();
$scripts->addFile('tbl_find_replace.js');
```

发现getRegplacePreview在 tbl_find_replace.php中使用，并且 find 与 replaceWith参数经POST方法进行传递。至此参数与函数溯源完毕。

前端查看该界面是phpmyadmin所提供的查找并替换数据表的功能/该功能时针对某一数据库中的数据表进行的查询功能：如图所示



其中查找的参数就是 find 替换为 的参数就是replaceWith；

```
<legend>Find and replace</legend>
查找:
<input type="text" value="" name="find" required="">
替换为:
<input type="text" value="" name="replaceWith" required="">
字段:
<select name="columnIndex"> ⊙ </select>
```

现在针对这两个的参数都寻找到了，就剩下 第三个参数了，继续寻找。
第三个参数为 row[0]首先看到这个参数为一数组，猜想是由SQL语句查询并返回的第一个值。

```
if (is_array($result)) {
    foreach ($result as $index=>$row) {
        $result[$index][1] = preg_replace(
            "/" . $find . "/",     //回溯find函数
            $replaceWith,
            $row[0]
```

回溯result参数

```
$result = $GLOBALS['dbi']->fetchResult($sql_query, 0);
```

这里涉及到一个DBI接口链接数据库的问题，先不去考察它。针对这个漏洞只需要定位使用到的sql查询语句并解析值就可以了。
回溯$Sql_query

```
$column = $this->_columnNames[$columnIndex];
$sql_query = "SELECT "
    . PMA_Util::backquote($column) . ","
    . " 1," // to add an extra column that will have replaced value
    . " COUNT(*)"
    . " FROM " . PMA_Util::backquote($this->_db)
    . "." . PMA_Util::backquote($this->_table)
    . " WHERE " . PMA_Util::backquote($column)
    . " RLIKE '" . PMA_Util::sqlAddSlashes($find) . "' COLLATE "
    . $charSet . "_bin"; // here we
    // change the collation of the 2nd operand to a case sensitive
    // binary collation to make sure that the comparison is case sensitive
$sql_query .= " GROUP BY " . PMA_Util::backquote($column)
    . " ORDER BY " . PMA_Util::backquote($column) . " ASC";

$result = $GLOBALS['dbi']->fetchResult($sql_query, 0);
```

SQL语句可理解为
Select $columnname ,1,cont(*) from database.table_name where $columnname rLike '$find' collate $charset_bin Group BY $columnname order by $column ASC;

并将这个查询后的值作为键值对，把键值对的第一个值给了 preg_replace()函数的第三个参数。

```
class PMA_TableSearch
{
```

```
public function __construct($db, $table, $searchType)
{
    $this->_db = $db;
    $this->_table = $table;
    $this->_searchType = $searchType;
    $this->_columnNames = array();
    $this->_columnNullFlags = array();
    $this->_columnTypes = array();
    $this->_columnCollations = array();
    $this->_geomColumnFlag = false;
    $this->_foreigners = array();
    // Loads table's information
    $this->_loadTableInfo();
}
```

该类的一个析构方法，在创建这个对象的同时执行该方法；

接着回溯，可以看到漏洞触发的 tbl_find_replace.php 中引用了这个 PMA_TableSearch类创建了 $table_search 对象;如图：

```
require_once 'libraries/common.inc.php';
require_once 'libraries/TableSearch.class.php';

$response = PMA_Response::getInstance();
$table_search = new PMA_TableSearch($db, $table, "rep
```

在这里将 db table 参数 赋值。

回溯这两个参数发现在 /libraries/common.inc.php 中存在定义

```
/**
 * current selected database
 * @global string $GLOBALS['db']
 */
PMA_setGlobalDbOrTable('db');    //这里设置的全局变量

/**
 * current selected table
 * @global string $GLOBALS['table']
 */
PMA_setGlobalDbOrTable('table');
```


全局寻找该函数可以发现通过REQUEST方法来接收变量并将其设置为全局变量。

这两个参数分别为 数据可和数据表，经分析发现，该漏洞触发点，是在一个数据库表中操作而实现的，所以说，exploit-db 中所提供的POC是先创建数据表与列名，然后在进行参数的传递，这里可以直接将这个db与table 直接作为参数所提交：如图：

创建的数据库为test 数据表为"prgpwn" 该表中的first列 的值为“0/e”，该值也就是通过 $sql_qury sql语句中查询得到的 $row[0]

其中find传递的参数中包含 %00 将后面的反斜杠给截断。

最终执行时效果类似于：

```php
<?php
echo preg_replace("/0/e","system('dir');","0/e");

?>
```


转自先知社区

推荐文章

1 新永恒之蓝？微软SMBv3高危漏洞（CVE-2020-0796）分析复现

2 重大漏洞预警：ubuntu最新版本存在本地提权漏洞（已有EXP）