# Magento 2.3.1 :组合拳拿远程代码执行权限

Hulk / 2019-07-08 09:01:00 / 浏览数 5027 安全技术 漏洞分析 顶(0) 踩(0)

---

文章来源：https://blog.ripstech.com/2019/magento-rce-via-xss/

---

## 0x01 前言

这篇文章我将向你展示如何结合HTML注入和Phar反序列化来攻破一个流行的电子商务管理系统，此漏洞存在于Magento <=2.3.1。攻击者利用漏洞组合链可接管Magento商店并且重定向支付页面。

## 0x02 概述

攻击者可以往Magento商店的后端注入任意储存型JavaScript Payload，一旦受害者接触到这些恶意代码，漏洞利用代码将会在受害者的浏览器后自动执行，并且可以反弹shell。漏洞演示：

https://blog.ripstech.com/videos/magento-unauthenticated-stored-xss.mp4

管理员登入到后端仪盘表后，注入的代码将自动运行并劫持管理员会话，然后利用后台的RCE漏洞接管整个服务器。攻击者可以破坏公司的财务运营，例如，重定向所有的支

漏洞的利用需要结合 Authorize.Net ，一个专为Visa及其他信用卡网上定制的支付模块，并且它可以内置在Magento中。需要注意，Authorize.Net并不对漏洞负责，而是Magento。判别目标Magento站点是

我们将漏洞链影响评级为高，因为攻击者无需很高明的技巧或者社交工程。Magento商店一年的成交额超过1550亿美元，攻击者可能会非常积极地开发利用该漏洞链。

## 0x03 影响

启用了Authorize.Net模块以及部分易受攻击的Magento版本：

| 分支 | 修补版本 | 可利用版本 |
| --- | --- | --- |
| 2.3 | 2.3.2 | <= 2.3.1 |
| 2.2 | 2.2.9 | <= 2.2.8 |
| 2.1 | 2.1.18 | <= 2.1.17 |

## 0x04 漏洞分析

### 0x04.1 储存型XSS

Magento有多种清理用户输入的方式。这里主要讲如何绕过`escapeHtmlWithLinks()`方法来造成储存型XSS。漏洞位置在取消产品订单的备注处。

在讨论所述方法之前，为了方便理解，先看看`escapeHTML()`函数，了解Magento的主要清理方法：

```
/**
* Escape string for HTML context.
*
* AllowedTags will not be escaped, except the following: script, img, embed,
* iframe, video, source, object, audio
*
* @param string|array $data
* @param array|null $allowedTags
* @return string|array
*/
public function escapeHtml($data, $allowedTags = null)
```

可以看到，`escapeHTML()`方法解析用户输入的`$data`，移除所有HTML标签，这里并没有指定第二个参数`$allowedTags`。如果用户不设置第二个参数，所有输入的字符

没什么可以针对`escapeHTML()`的好方法，接下来看看经`escapeHTML()`处理前的代码会发生什么，此时被修改的数据最可能出现漏洞。这里我发现了`escapeHtmlWithL`

`escapeHtmlWithLinks()`方法用于移除白名单外的HTML标签。和`escapeHTML()`不同，它会移除所有属性除了`<a>`标签的`<href>`，目的是做到极致安全 :)。

`escapeHtmlWithLinks()`把`<a>`标签和用户输入解析到一个数组(`$matches`)中，代码位置：vendor/magento/module-sales/Helper/Admin.php

```
public function escapeHtmlWithLinks($data, $allowedTags = null)
{
   ■
   $data = str_replace('%', '%%', $data);
   $regexp = "#(?J)<a"
   ."(?:(?:\s+(?:(?:href\s*=\s*(['\"])(?<link>.*?)\\1\s*)|(?:\S+\s*=\s*(['\"])(.*?)\\3)\s*)*)|>)"
   .">?(?:(?:(?<text>.*?)(?:<\/a\s*>?|(?=<\w))|(?<text>.*)))#si";
```

```
while (preg_match($regexp, $data, $matches)) {
■
```

下一步代码会创建一个简易标签来清理链接文本和href属性中包含的URL。

清理后的链接作为数组储存在$links数组，稍后还会用到。escapeHtmlWithLinks()会抛掉清理后的<a>标签，使用%$is代替用户输入的字符串，其中$i是数字代表<a

■

```
while (preg_match($regexp, $data, $matches)) {
$text = '';
if (!empty($matches['text'])) {
    $text = str_replace('%%', '%', $matches['text']);
}
$url = $this->filterUrl($matches['link'] ?? '');
//Recreate a minimalistic secure a tag
$links[] = sprintf(
    '<a href="%s">%s</a>',
    htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false),
    $this->escaper->escapeHtml($text)
);
$data = str_replace($matches[0], '%' . $i . '$s', $data);
++$i;
}
```

大致的方法就是这样，为了有更直观的体验，我举个例子：

Hello, <a href="/the-world/" title="Hello World">World!</a>将会变为Hello, %1s

escapeHtmlWithLinks()方法替换用户输入中的<a>为%s后，将结果传递给escapeHTML()。这安全地清理掉用户输入的有害字符。然而，代码会将清理结果return给v

■

```
} // End of while
$data = $this->escaper->escapeHtml($data, $allowedTags);
return vsprintf($data, $links);
```

vsprintf — 返回格式化字符串

```
vsprintf ( string $format , array $args ) : string
```

这是XSS漏洞的根源，让我们来看看XSS Payload的处理过程。

| 步骤 | 用户输入字符串 |
| --- | --- |
| 从用户输入字符串解析<a>标签 | <i id=" <a href='http://onmouseover=alert(/XSS/)'>a link</a> "> a malicious link  |
| 将<a>标签替换为%1s |  a malicious link  |
| 清理其他额外的标签 |  a malicious link  |
| 把清理后的<a>插入到已清理的其他字符串中 | <i id=" <a href="http://onmouseover=alert(/XSS/)>">a link</a> "> a malicious link  |

从上表可以看出，<a>标签被替换为%1s然后清理用户输入的其他字符。因为%1s属于安全字符，它被标记为安全的。最后escapeHtmlWithLinks()方法末段的vsprintf

攻击者可以利用这点来注入任意代码到结果字符中。通过注入恶意onmouseover等事件句柄和style属性可以使链接不可见，只要受害者访问页面并移动鼠标，Payload就

escapeHtmlWithLinks()方法用于清理用户取消订单中输入的备注，订单经由Authorize.Net处理。上过上面的绕过措施，攻击者可以注入任意JavaScript代码到订单
Payload就触发了。

0x04.2 Phar反序列化

只要拿下管理员权限后，攻击者就可以滥用所见即所得（WYSIWYG）富文本编辑器中负责图片处理的控制器来执行Phar
反序列化。下面这段代码展示了POST参数__directive的内容如何传递给image Adapter
类的open()方法。该方法在内部将用户输入传递给函数getimagesize()，而这个函数易受Phar反序列化的影响。

```
public function execute()
    {
        $directive = $this->getRequest()->getParam('___directive');
        $directive = $this->urlDecoder->decode($directive);
        ■
        $image = $this->_objectManager->get(\Magento\Framework\Image\AdapterFactory::class)->create();
        try {
            $image->open($imagePath);
        ■
```

将phar://流包装器注入到图片处理程序中，触发PHP反序列化，最终导致远程代码执行。

0x05 时间线

| 日期 | |
|---|---|
| 2018/9/25 | 上报Magento 2.2.6中存在一个存储的XSS漏洞。 |
| 2018/11/28 | Magento推出补丁程序2.2.6和2.1.16 |
| 2018/12/13 | 报告了Magento 2.3.0中的绕过方法。 |
| 2019/1/11 | 向Magento安全团队报告了Phar反序列化漏洞。 |
| 2019/1/26 | 我们发现存储型XSS在具有特定配置的Magento上可被用户利用，并通知Magento。 |
| 2019/1/29 | Magento验证了存在漏洞。 |
| 2019/3/26 | Magento推出补丁程序2.3.1,<br>2.2.8和2.1.17。日志显示Phar反序列化漏洞已修复，未提到XSS漏洞。 |
| 2019/4/09 | Magento把XSS漏洞状态标记为“已解决” |
| 2019/4/09 | 我们询问Magento是否已修复漏洞，因为更新日志没有提到它，并且没有修改escapeHTM |
| 2019/4/10 | Magento重新标记漏洞状态。 |
| 2019/6/25 | Magento推出补丁程序 2.3.2, 2.2.9和2.1.18。 |

0x06 总结

本文介绍了储存型XSS和Phar反序列化漏洞，利用它们攻击者可以大规模攻击Magento站点。从漏洞分析可以看出，现今主要的安全漏洞的根源都是代码层面上多重清理，



上一篇：0ctf 反序列化逃逸复现 下一篇：代码审计 xxxdisk前台Get...

1. 0 条回复
   - 动动手指，沙发就是你的了！