XX博客平台

**Web 应用程序渗透测试代码审查**

测试人：陈煜翔

学号：20178314057

**目录**

1. [介绍 3](#_TOC_250022)
   1. [完整描述 3](#_TOC_250021)
2. [关键词一览 4](#_TOC_250020)
   1. [XSS 4](#_TOC_250019)
   2. [拒绝服务攻击 4](#_TOC_250018)
   3. [权限问题 4](#_TOC_250017)
   4. [其他 5](#_TOC_250016)
3. [渗透过程 6](#_TOC_250015)
   1. [XSS 6](#_TOC_250014)
      1. [博客用户相关的XSS攻击 6](#_TOC_250013)
      2. 导航栏的XSS攻击
   2. [拒绝服务攻击 11](#_TOC_250011)
      1. [不受控制的资源消耗 11](#_TOC_250010)
   3. [权限问题 13](#_TOC_250009)
      1. [权限丢失 13](#_TOC_250008)
      2. [权限绕过 15](#_TOC_250007)
      3. [越权漏洞 17](#_TOC_250006)
   4. [其他 19](#_TOC_250005)
      1. [不安全的Token使用 19](#_TOC_250004)
   5. [相关 20](#_TOC_250003)
4. [附录 21](#_TOC_250002)
   1. [工具 21](#_TOC_250001)

# 介绍

XX博客系统正在成为一个广泛的博客平台。随着越来越多的用户离开Wordpress，接受更多内容这类小众的博客平台，这些平台专注于写作和阅读。在这一点，本系统是最常用的软件之一，因为它得到了很多的关注，我们开始担心其用户的安全性

以下所做的测试是由个人进行的，均处在虚拟环境中，没有任何实体网站受到损害。

对于代码审阅部分，使用从 GitHub下载的应用程序源代码的本地副本。

对于Web应用程序渗透测试，使用自己的服务器。

在以下章节中，您将找到测试期间和每个测试中发现的漏洞的描述：

* + 危险漏洞评估表
  + 漏洞利用说明
  + 漏洞源码错误部分
  + 截图

# 关键词

在本章中，我将列出在测试中发现的所有漏洞，稍后我将逐一详细讨论。

## XSS

XSS 允许攻击者注入代码客户端，然后在受害者计算机上执行。

存储和反射的 XSS 有两种不同类型的 XSS。

第一种类型是最危险的，因为注入代码（例如 Javascript）存储在网页上，并且将由所有用户触发。

相反，第二种类型要求受害者自己注入代码，并且它不存储在网页上。用户离开页面后，代码将被删除。

在 本次测试的博客系统 中，我们发现**三个**存储的 XSS，例如，这可能导致会话劫持。

### XSS 在博客的标志和封面

### XSS 在用户的个人页面

### XSS在标签管理器

## 拒绝服务攻击

DoS 漏洞允许攻击者在目标服务器中创建故障。最常见的问题是，带宽大于服务器的攻击者试图to 超载其网络容量，拒绝对真实用户的访问。

在我们的例子中，漏洞在应用程序中，它可能导致服务器崩溃。

### DoS，文件系统中的不受控制的资源消耗。

## 权限问题

特权问题可能导致多个问题，例如，特权升级用于增加用户的权限，例如使其成为管理员。

在本系统中，发现了一个权限减少，这可能导致拒绝服务，以及几个特权绕过，我稍后将讨论。

### 权限减少，可能导致特权升级

### 读取存储的权限绕过

### 发布帖子中的权限升级

## 其他

我将在此处列出未包含在前几个类别中的漏洞。

我们发现，软件用于保持用户身份验证的令牌不是存储安全性，如果与 XSS 漏洞结合使用，则可能导致会话劫持。

### 不安全的令牌存储

# 漏洞详情信息

## XSS

## XSS在博客/用户的图像

在博客的图片、博客封面、用户的头像和用户封面上都发现了存储的XSS。

这可能导致任意执行代码客户端，如 javascript。

请注意，只有旧的浏览器版本是易受攻击的。此外，要利用它在博客的图片和博客的封面上，用户必须作为管理员或所有者进行身份验证，而对于用户的头像和封面，它可能只是一个作者。

下表总结了漏洞的严重性。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **LOW** | **Technical Impact** | | **LOW** | | **Business Impact** | | **LOW** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 3 4 | | | 3 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **4.0** | | | | | |

**存储在博客图像**中的 XSS

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **LOW** | **Technical Impact** | | **LOW** | | **Business Impact** | | **LOW** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 4 4 | | | 4 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **4.5** | | | | | |

以下代码段显示了导致上述漏洞的原因。

updateConfigTheme = function () { config.set({

theme: {

title: (settingsCache.title && settingsCache.title.value) || '',

description: (settingsCache.description && settingsCache.description.value) || '', logo: (settingsCache.logo && settingsCache.logo.value) || '',

cover: (settingsCache.cover && settingsCache.cover.value) || ''

}

});

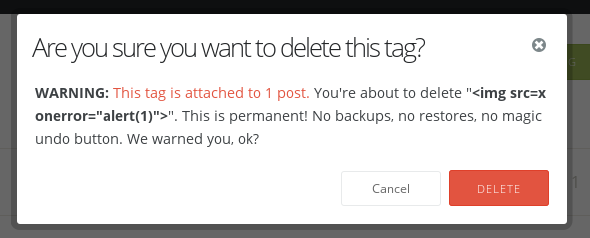
};

## 选项卡管理器中的 XSS

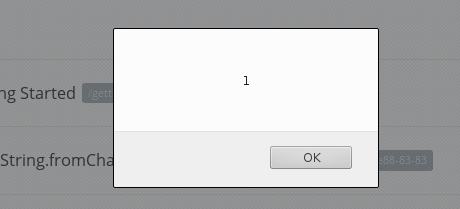
在标签管理器中找到了其他存储的 XSS;任何用户（作者、编辑、管理员或所有者）都可以创建新帖子，并例如将 javascript 代码作为帖子的标记。当某人（如管理员）注意到该代码并继续删除它时，将触发 javascript 代码，例如，管理员令牌被盗并劫持其会话。

下表总结了 Vulnerabilitie 的严重程度。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **LOW** | **Technical Impact** | | **MEDIO** | | **Business Impact** | | **LOW** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 4 6 | | | 7 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
|  | | | | **5.8** | | | | | |



**恶意标记的标记**



**Javascript 警报**



**使用 JS 注入的请求和响应**



**嵌入 JS 的源代码**

以下代码段显示了导致上述漏洞的原因

confirmAccept: function () { var tag = this.get('model'),

name = tag.get('name'), self = this;

this.send('closeSettingsMenu'); tag.destroyRecord().then(function () {

self.notifications.showSuccess('Deleted ' + name);

}).catch(function (error) { self.notifications.showAPIError(error);

});

},

## 拒绝服务

## 不受控制的资源消耗

在平台中发现了一个重要漏洞。这将导致拒绝服务，并可能导致服务器崩溃。它允许经过身份验证的攻击者、作者、编辑、管理员或所有者耗尽文件系统空间。这并不重要。

这是因为当用户更新/更改其头像或覆盖前一个头像时不会被删除，而且不会对图像大小进行控制。

下表总结了漏洞的严重性。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **LOW** | **Technical Impact** | | **MEDIUM** | | **Business Impact** | | **LOW** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 4 7 | | | 7 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **5.7** | | | | | |

以下代码段显示了导致上述漏洞的原因。正如您所看到的，前一个图像的"删除"全部完成客户端

UploadUi = function ($dropzone, settings) {

var $url = '<div class="js-url"><input class="url js-upload-url" type="url" placeholder="http://"/></div>',

$cancel = '<a class="image-cancel js-cancel" title="Delete"><span class="hidden">Delete</span></a>',

$progress = $('<div />', {

class: 'js-upload-progress progress progress-success active', role: 'progressbar',

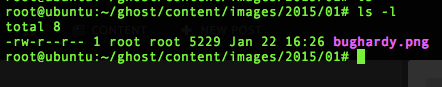
'aria-valuemin': '0',

'aria-valuemax': '100'

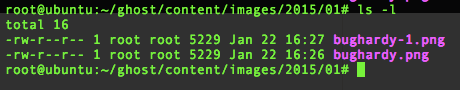
}).append($('<div />', {

class: 'js-upload-progress-bar bar', style: 'width:0%'

}));



**上传的第一个图像**



**第二个图像上传，第一个图像仍然存在**

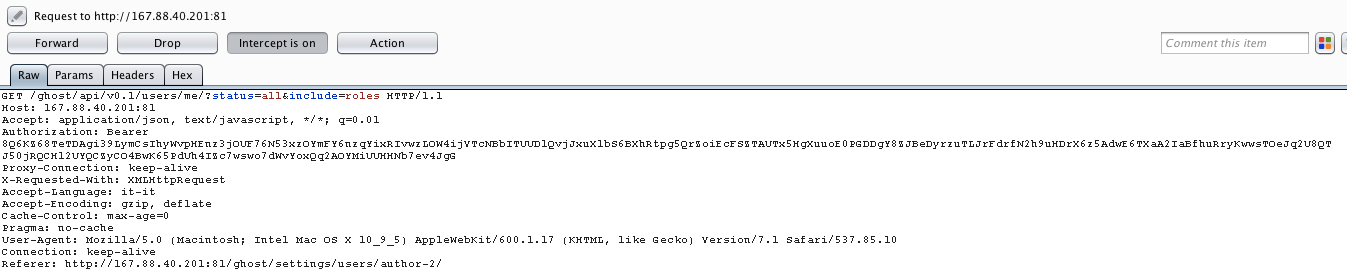
## 权限问题

## 减少特权

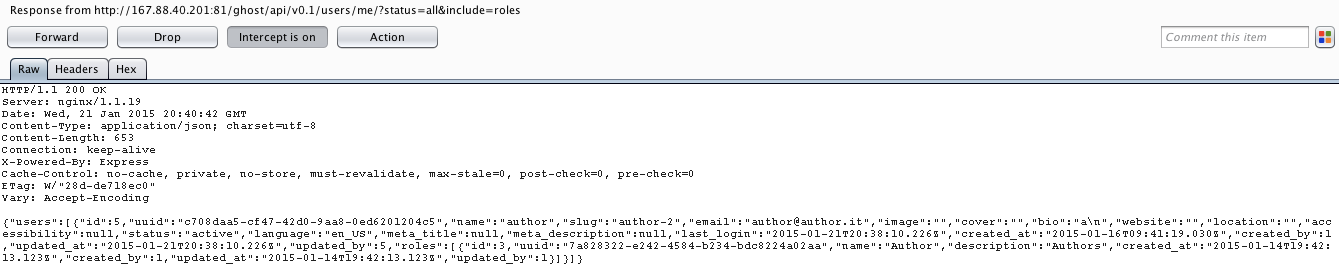
减少特权是一种旨在减少其他用户权限的漏洞。这样做有几个原因，例如造成中断;假设作者可以删除所有管理员和编辑，然后谁可以缓和他的帖子？这种权限减少以非常有趣的方式导致特权升级。假设A是编辑，B 是博客的所有者。现在，C想成为管理员，现在可以将 B 的权限减少到管理员，在执行此操作时，他还可以更改目标的个人信息，例如将 B 的电子邮件更改为自己的电子邮件（请注意：要更改具有更多权限的其他用户的详细信息，您必须减少它们并更改同一请求中的详细信息）。然后重置 B 的密码，他将收到电子邮件，现在可以更改 B 的密码并登录到他的帐户。

您应该注意，作者无法将其他用户的角色更改为超过级别 3 的内容，即"作者"。也就是说，你可以在这里看到另一个问题;作者可以编辑任何其他作者的详细信息，例如更改他们的电子邮件，或将其帐户警告级别设置为"锁定"，或重置警告级别以能够强制其密码。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **MEDIUM** | **Technical Impact** | | **HIGH** | | **Business Impact** | | **MEDIUM** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 4 8 | | | 8 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **6.8** | | | | | |



**用户信息请求**



**用户降级和电子邮件更改的请求和响应**

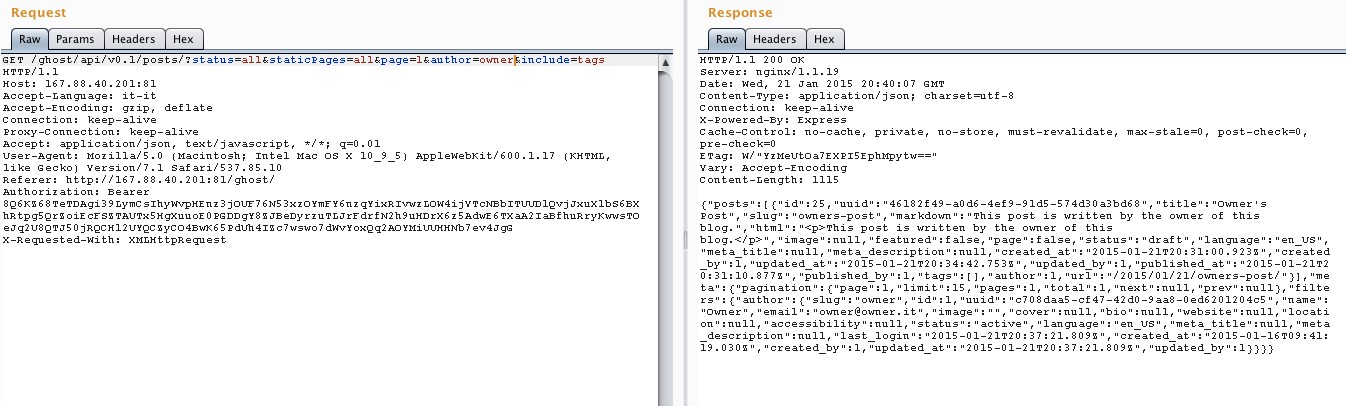
**Credits: Matteo Beccaro**

## 特权绕过

此漏洞允许任何用户读取任何用户的草稿。这是一个权限绕过，因为它被认为是不正确的，作者可以阅读所有所有者草稿，例如。

这是用户当前草稿请求期间未选中的参数;我们可以更改此参数以欺骗我们作为任何其他用户，我们只需要知道它的"slug"，这不是机密信息，因为是已知的，也是未经身份验证的用户.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **MEDIUM** | **Technical Impact** | | **LOW** | | **Business Impact** | | **LOW** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 7 4 4 | | | 7 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **4.7** | | | | | |

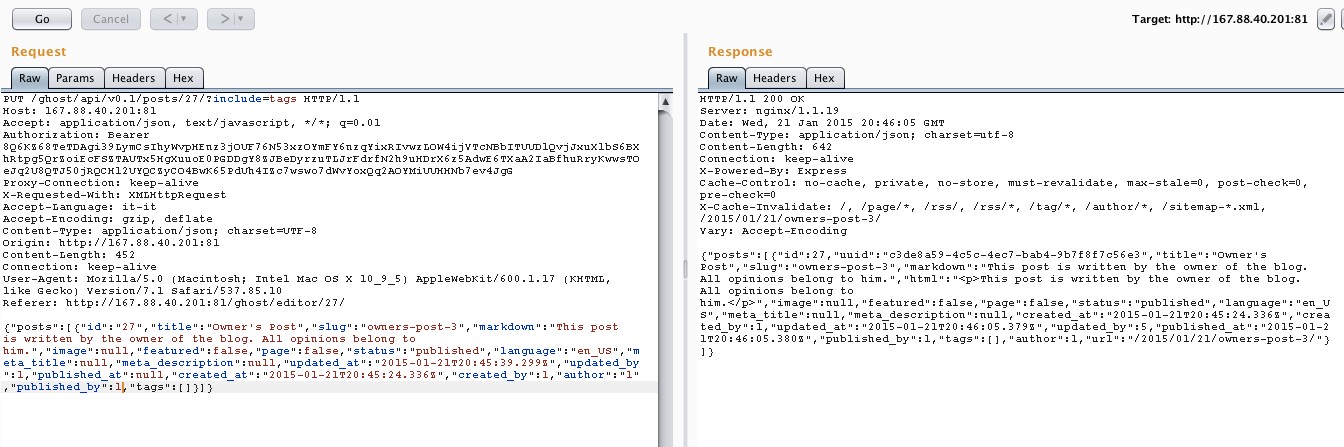


**请求我们欺骗我们的 ID 以便读取所有者的草稿.**

## 权限升级

系统中的一个重要缺陷是，用户可能会欺骗其身份，并发布以其他用户的名义发布文章。这意味着作者可以以所有者的名义发布文章，例如;然后没有办法说谁真正发表了那篇文章。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **MEDIUM** | **Technical Impact** | | **MEDIUM** | | **Business Impact** | | **MEDIUM** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 5 4 7 | | | 8 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **6.5** | | | | | |



**作者以博客所有者的名义发表文章的请求和响应。**



**博客文章以所有者为作者发布。**

## 其他

## 不安全的令牌存储

在系统 中，我们发现承载令牌不是存储为 Cookie 或具有可防止其被盗的认证信息，而是在它中仅存储在浏览器的*本地存储*中;这在 XSS 漏洞中可能导致会话劫持。令牌应放入启用 HttpOnly 标志的 Cookie 中，以防止此类问题.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Likelihood** | | **MEDIUM** | **Technical Impact** | | **MEDIUM** | | **Business Impact** | | **MEDIUM** |
| **Base Score** | | | | **Temporal Modificator** | | | | | |
| **Access Vector** | **Access Authentication Impact Complexity** | | | **Exploitability** | | **Remediation Level** | | **Awareness** | |
| 6 | 4 4 7 | | | 7 | | NdA | | NdA | |
| **Envirovment Modificator** | | | | **Overall Score** | | | | | |
| **Collateral Damage** | **Target Distribution Requirement** | | | Final Score | | | | | |
| NdA | NdA NdA | | | **5.5** | | | | | |



**对本地存储令牌的 Javascript 访问**

## 相关

在这里，我想指出两个可能的漏洞;第一个不是由开发团段团队直接造成的，而对于第二个团队，团队不确定其性质，如果它是一个功能与否。

第一个问题是使用 bcrypt比较函数来检查用户提供的密码哈希是否与存储在数据库中的哈希值相同。

if (user.get('status') !== 'locked') {

return bcryptCompare(object.password, user.get('password')).then(function (matched) { if (!matched) {

…

This function is vulnerable to **timing attack** and should then be avoided.

for (var i = 0; i < max\_length; ++i) {

if (hash\_data\_length >= i && encrypted\_length >= i && hash\_data[i] != encrypted[i]) { same = false;

}

}

第二个问题反而在系统内。任何用户（作者、编辑人员、管理员和所有者）都可以将 javascript 代码注入文章中。这可能导致很容易利用的 XSS;和以前我们指出的漏洞可能会导致一些损害。但团队不确定这是 Ghost 团队希望保留的功能，还是将来会替换或删除。

# 附录

## 工具

多种工具执行开源和专有测试。

* + - Burp Proxy
    - Fiddler
    - Tamper Data Firefox extension
    - Python
    - Curl
    - ZAP Proxy