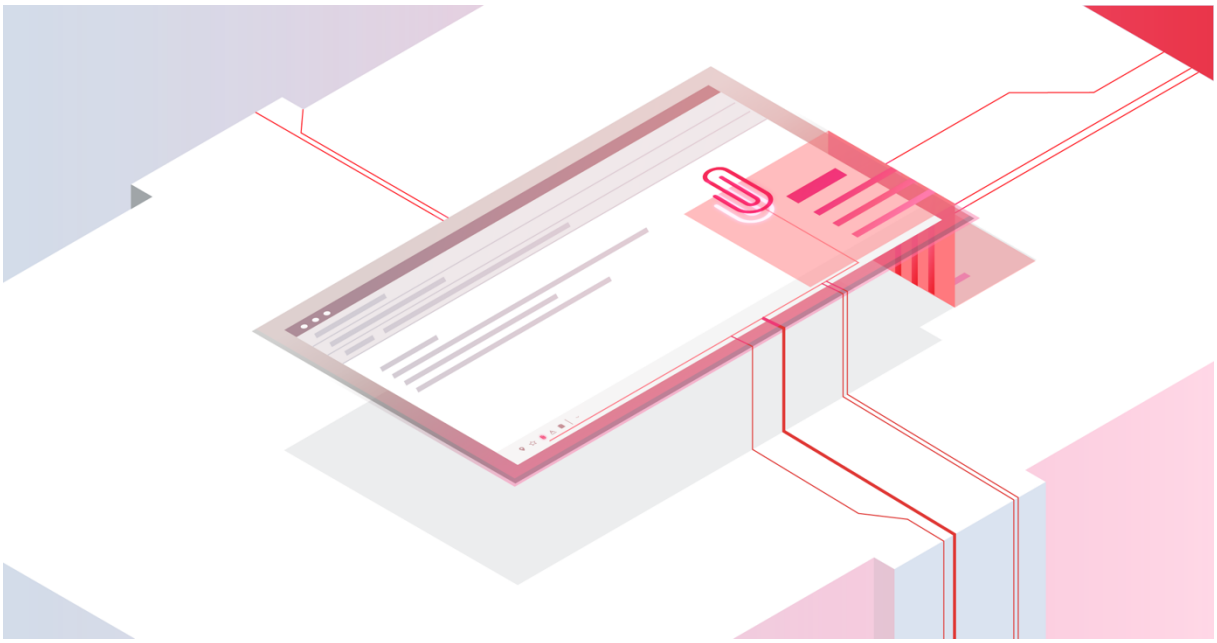
sonarsource

Horde Webmail 5.2.22 - Account Takeover via Email

BY SIMON SCANNELL| FEBRUARY 22, 2022



Horde Webmail is a free, enterprise-ready, and browser-based communication suite developed by the Horde project. It is a popular webmail solution for universities and government agencies to exchange sensitive email messages on a daily basis. It is also shipped as part of the popular hosting solution cPanel that is used by many enterprises to manage their website.

We discovered a code vulnerability in Horde that allows an attacker to gain full access to the email account of a victim when it loads the preview of a harmless-looking email attachment. This gives the attacker access to all sensitive and perhaps

secret information a victim has stored in their email account and could allow them to gain further access to the internal services of an organization.

Although we reported this vulnerability almost 6 months ago, there is currently no official patch available. Hence, we provide recommendations on how to mitigate this code vulnerability at the end of this blog post. This can be done easily by disabling the affected feature, which does not have a big impact on the usability of the software. By releasing the vulnerability and patch details, we hope to raise visibility and to enable administrators to secure their servers.

Impact

This Stored XSS vulnerability was introduced with the commit 325a7ae, 9 years ago. It likely affects all the Horde instances deployed as of today and works under default configurations.

An attacker can craft an OpenOffice document that when transformed to XHTML by Horde for preview can execute a malicious JavaScript payload. The vulnerability triggers when a targeted user views an attached OpenOffice document in the browser. As a result, an attacker can steal all emails the victim has sent and received.

By default, Horde ships with an admin panel, which allows admins to execute arbitrary system commands on a Horde instance through the administrative interface. If an attacker succeeds in targeting an administrator with a personalized, malicious email, they could abuse this privileged access to take over the entire webmail server.

Technical Details

In the following sections, we go into detail about an unusual XSS vulnerability that occurs due to a relaxed matching rule in an XSLT document.

Background: OpenOffice documents and XSLT transformations

An OpenOffice document is a ZIP file containing XML documents, as well as other files needed to render a document, such as images. When Horde is asked to convert

an OpenOffice document to HTML for its previsualization, it uses XSLT (eXtensible Stylesheet Language Transformations) to convert the XML files contained within the OpenOffice document.

XSLT documents are XML documents containing directives that instruct an XSLT processor on how to convert an input XML document into HTML markup. Let's learn about some directives to gain a better understanding of XSLT and the root cause of the vulnerability discussed in this blog post.

The following snippet shows the declaration of an XSL stylesheet and uses the `<xsl:output>` directive to declare that `html` code will be produced by this stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:output method="html" />
...
```

Another important directive is the `<xsl:template>` directive. In the following example, the template is always processed as it is run when matched against the root element of an XML document:

```
<xsl:template match="/">
<html>
   <body>
       <h1>
           <xsl:value-of select="/BlogPost/Title"/>
       </h1>
   </body>
</html>
</xsl:template>
```

In the example of the `<xsl:template>` above, the `<html>`, `<body>` and `<h1>` tags are not processed and become a part of the final HTML document that is produced. However, the value of the `<h1>` tag is dynamically generated. An XPath query is used to select the value of a Title element within the input XML document to be rendered. This is done with the `<xsl:value-of select="/BlogPost/Title"/>` directive.

Let's assume the following XML document, which holds information about a blog post, was to be rendered:

```
<?xml version="1.0"?>
<BlogPost>
 <Title>A Blog Post about XSLT vulnerabilities</Title>
 <Authors>
```

```
    <Author>Foo</Author>
    <Author>Bar</Author>
  </Authors>
  <Content>Lorem Ipsum</Content>
</BlogPost>
```

In this case, the resulting HTML would look like the following, as the value of the `<h1>` tag is dynamically produced:

```
<html>
    <body>
        <h1>
            A Blog Post about XSLT vulnerabilities
        </h1>
    </body>
</html>
```

With this background knowledge in mind, let's look at how an XSS vulnerability could arise when translating XML documents into HTML.

Stored XSS vulnerability in crafted OpenOffice document

When Horde is asked to render an OpenOffice document for a user, it utilizes the `opendoc2xhtml.xsl` stylesheet file developed by the OpenOffice project. The following code snippet shows how this XSL document is loaded and then used to transform the attacker-controlled `content.xml` file:

Horde/Mime/Viewer/Ooo.php

```
        $xslt = new XSLTProcessor();
        $xsl = new DOMDocument();
        $xsl->load(realpath(__DIR__ .
'/Ooo/export/xhtml/opendoc2xhtml.xsl'));
        $xslt->importStylesheet($xsl);
        $xslt->setParameter('http://www.w3.org/1999/XSL/Transform', array(
            'metaFileURL' => 'file://' . $tmpdir . 'meta.xml',
            'stylesFileURL' => 'file://' . $tmpdir . 'styles.xml',
            'java' => false,
        ));
        $xml = new DOMDocument();
        $xml->load(realpath($tmpdir . 'content.xml'));
        $result = $xslt->transformToXml($xml);
        if (!$result) {
            $result = libxml_get_last_error()->message;
        }

        return array(
            $this->_mimepart->getMimeId() => array(
                'data' => $result,
                'status' => array(),
                'type' => 'text/html; charset=UTF-8'
            )
        );
    }
```

Once the XML file has been converted, it is simply returned to the user **without** any further sanitization after the conversion from OpenOffice document to XHTML. This

means that if an attacker could craft an OpenOffice document that leads to JavaScript injection in the resulting XHTML, then a XSS vulnerability occurs.

Finding an injection point in an XSLT stylesheet

In a previous example, we used the `<xsl:value-of>` directive to query the value of a user-controlled XML element and embed it into outputted HTML code. By default, this directive escapes its output and thus won't allow XSS. The same escaping applies to attributes, which means we can't inject attributes into HTML elements.

The vulnerability we discovered comes from an injection point where no escaping is applied, which is shown in the following snippet:

```
<xsl:template match="draw:object[math:math]">
   <math xmlns="http://www.w3.org/1998/Math/MathML">
      <xsl:apply-templates select="math:math/math:semantics/*"
mode="math"/>
   </math>
</xsl:template>
```

The logic of it could be read as: for all `<draw:object>` elements which have a `<math:math>` element, apply all templates that have their mode attribute set to math. These templates should operate on all children of a `<math:semantics>` child. This is because of the `*` in the XPath query.

In practice, this means that this template is executed when the attacker-controlled OpenOffice document contains the following tags:

```
<draw:object><math:math><math:semantics>...</math:semantics></math:math></d
raw:object>
```

The following template operates on any child of the previously shown elements:

```
<xsl:template match="*" mode="math">
   <xsl:element name="{local-name()}"
namespace="http://www.w3.org/1998/Math/MathML">
      <xsl:apply-templates select="@*|node()" mode="math"/>
   </xsl:element>
</xsl:template>
```

It uses the `<xsl:elements>` directive to dynamically create a new HTML tag. The name of the tag becomes the tag that is currently operated on, which is determined by the `local-name()` function. As the parent template passed any element to this

template, as instructed by the `*` in the XPath query, an attacker can create arbitrary HTML elements, including `<script>` tags.

The ability to create arbitrary HTML tags leads to the ability of an attacker to craft an OpenOffice document containing the following markup:

```
<draw:object><math:math><math:semantics><p>XSS payload:
</p><script>alert('xss');</script>
```

When a victim then views such an OpenOffice document attachment, the XSS payload triggers and gives an attacker full access to their session. This means the attacker can steal all emails and, in a worst-case scenario, even execute arbitrary system commands if the victim has the administrator role.

Patch

As there is no official patch available at the time of writing, we recommend disabling the rendering of OpenOffice attachments. To do so, administrators can edit the `config/mime_drivers.php` file in the content root of their Horde installation. As shown in the snippet below, add the `'disable' => true` configuration option to the OpenOffice mime handler:

```
/* OpenOffice.org/StarOffice document display. */
'ooo' => array(
   'disable' => true,       // <---- HERE
   'handles' => array(
       'application/vnd.stardivision.calc',
       'application/vnd.stardivision.draw',

       // ...
```

Users will still be able to download the OpenOffice documents and view them locally, but Horde won't attempt to render it in the browser. With this, the vulnerable feature is not used and the Horde instance is protected against exploitation of this vulnerability.

Timeline

Date	Action
2021-08-26	We report the XSS issue to the vendor and inform them of our 90-day disclosure policy

Date	Action
2021-08-31	The vendor confirms the vulnerability
2021-09-23	We ask the Vendor for a status update (no reply)
2021-10-28	We ask the Vendor for a status update (no reply)
2022-02-17	We inform the vendor of the upcoming release

Summary

In this blog post, we presented an unusual XSS vulnerability in the Horde webmailer. The vulnerability allows an attacker to craft a malicious OpenOffice document that, when previewed as an email attachment, enables an attacker to steal all emails from the victim. Since there is no official patch available yet, we highly recommend all Horde users to disable the affected feature as described in this blog post.

In general, we recommend developers to always sanitize HTML documents after they have been produced by XSLT rendering, especially when the conversion is performed by a third party library or stylesheet. The most modern way of doing this would be to use a library such as DOMPurify to ensure that only secure HTML elements are produced by the OpenOffice document.