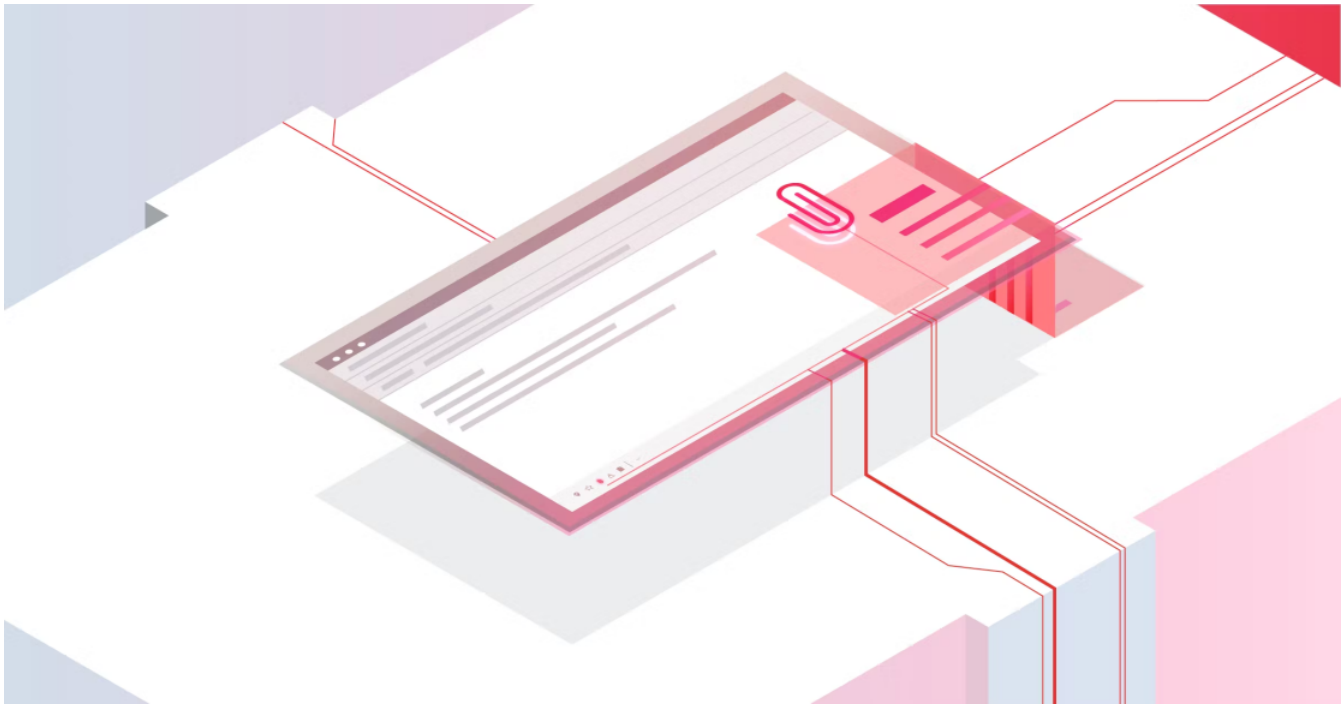
Horde Webmail - Remote Code Execution via Email

BY SIMON SCANNELL | MAY 31, 2022

Security



A webmail application enables organizations to host a centralized, browser-based email client for their members. Typically, users log into the webmail server with their email credentials, then the webmail server acts as a proxy to the organization's email server and allows authenticated users to view and send emails.

With so much trust being placed into webmail servers, they naturally become a highly interesting target for attackers. If a sophisticated adversary could compromise a webmail server, they can intercept every sent and received email, access password-reset links, and sensitive documents, impersonate personnel and steal all credentials of users logging into the webmail service.

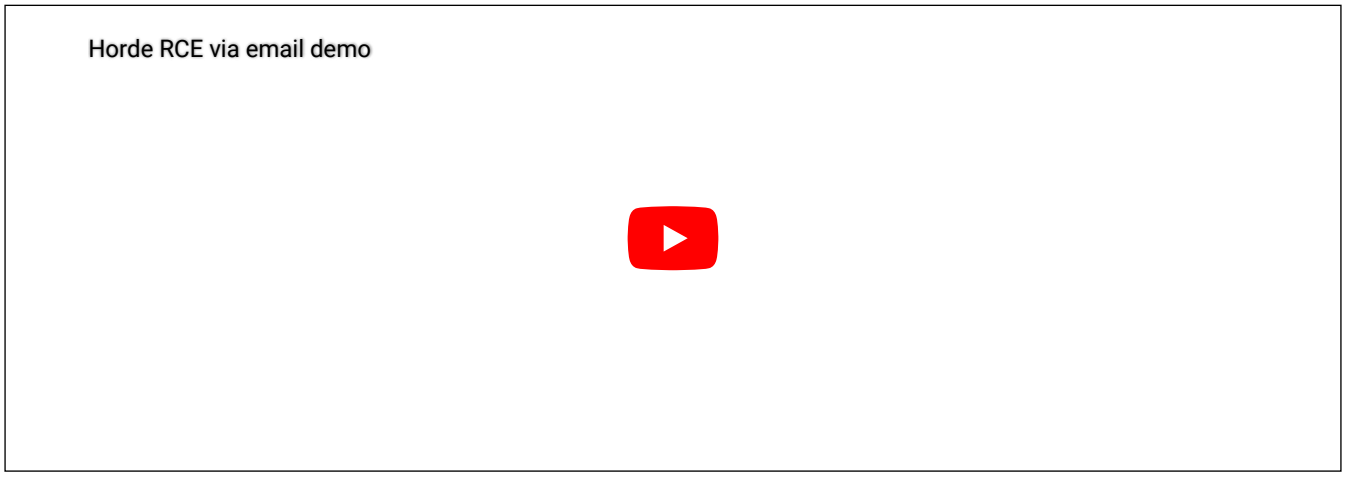
This blog post discusses a vulnerability that the Sonar R&D team discovered in Horde Webmail. The vulnerability allows an attacker to fully take over an instance as soon as a victim opens an email the attacker sent. At the time of writing, no official patch is available.

Impact

iser of a Horde instance to execute arbitrary

The vulnerability exists in the default configuration and can be exploited with no knowledge of a targeted Horde instance. We confirmed that it exists in the latest version. The vendor has not released a patch at the time of writing.

Another side-effect of this vulnerability is that the clear-text credentials of the victim triggering the exploit are leaked to the attacker. The adversary could then use them to gain access to even more services of an organization. This is demonstrated in our video:



Technical details

In the following sections, we go into detail about the root cause of this vulnerability and how attackers could exploit it.

Background - Horde Address Book configuration

Horde Webmail allows users to manage contacts. From the web interface, they can add, delete and search contacts. Administrators can configure where these contacts should be stored and create multiple address books, each backed by a different backend server and protocol.

The following snippet is an excerpt from the default address book configuration file and shows the default configuration for an LDAP backend:

turba/config/backends.php

```
$cfgSources['personal_ldap'] = array(
   // Disabled by default
   'disabled' => true,
   'title' => _("My Address Book"),
   'type' => 'LDAP',
   'params' => array(
       'server' => 'localhost',
       'tls' => false,
    // …
```

As can be seen, this LDAP configuration is added to an array of available address book backends stored in the `$cfgSources` nfigure the LDAP driver.

The following code snippet demonstrates typical usage of this pattern:

turba/merge.php

```
14 require_once __DIR__ . '/lib/Application.php';
15 Horde_Registry::appInit('turba');
16
17 $source = Horde_Util::getFormData('source');
18 // …
19 $mergeInto = Horde_Util::getFormData('merge_into');
20 $driver = $injector->getInstance('Turba_Factory_Driver')->create($source);
21 // …
30 $contact = $driver->getObject($mergeInto);
```

The code snippet above shows how the parameter `$source` is received and passed to the `create()` method of the `Turba_Factory_Driver`. Turba is the name of the address book component of Horde.

Things start to become interesting when looking at the `create()` method:

turba/lib/Factory/Driver.php

```
51     public function create($name, $name2 = '', $cfgSources = array())
52     {
53     // …
57         if (is_array($name)) {
58             ksort($name);
59             $key = md5(serialize($name));
60             $srcName = $name2;
61             $srcConfig = $name;
62         } else {
63             $key = $name;
64             $srcName = $name;
65             if (empty($cfgSources[$name])) {
66                 throw new Turba_Exception(sprintf(_("The address book \"%s\" does not exist."), $name));
67             }
68             $srcConfig = $cfgSources[$name];
69         }
```

On line 57, the type of the `$name` parameter is checked. This parameter corresponds to the previously shown `$source` parameter. If it is an array, it is used directly as a config by setting it to `$srcConfig` variable. If it is a string, the global `$cfgSources` is accessed with it and the corresponding configuration is fetched.

This behavior is interesting to an attacker as Horde expects a well-behaved user to send a string, which then leads to a trusted configuration being used. However, there is no type checking in place which could stop an attacker from sending an array as a parameter and supplying an entirely controlled configuration.

Some lines of code later, the `create()` method dynamically instantiates a driver class using values from the attacker-controlled array:

With this level of control, an attacker can choose to instantiate an arbitrary address book driver and has full control over the parameters passed to it, such as for example the host, username, password, file paths etc.

Instantiating a driver that enables an attacker to execute arbitrary code

The next step for an attacker would be to inject a driver configuration that enables them to execute arbitrary code on the Horde instance they are targeting.

We discovered that Horde supports connecting to an IMSP server, which uses a protocol that was drafted in 1995 but never finalized as it was superseded by the ACAP protocol. When connecting to this server, Horde fetches various entries. Some of these entries are interpreted as PHP serialized objects and are then unserialized.

The following code excerpt from the `_read()` method of the IMSP driver class shows how the existence of a `__members` entry is checked. If it exists, it is deserialized:

turba/lib/Driver/Imsp.php

```
223   if (!empty($temp['__members'])) {
224       $tmembers = @unserialize($temp['__members']);
225   }
```

Due to the presence of viable PHP Object Injection gadgets discovered by Steven Seeley, an attacker can force Horde to deserialize malicious objects that lead to arbitrary code execution.

Exploiting the vulnerability via CSRF

By default, Horde blocks any images in HTML emails that don't have a `data:` URI. An attacker can bypass this restriction by using the HTML tags `<picture>` and `<source>`. A `<picture>` tag allows developers to specify multiple image sources that are loaded depending on the dimensions of the user visiting the site. The following example bypasses the blocking of external images:

```
<picture>
  <source media="(min-width:100px)" srcset="../../?EXPLOIT">
  <img src="blocked.jpg" alt="Exploit image" style="width:auto;">
</picture>
```

Patch

At the time of writing, no official patch is available. As Horde seems to be no longer actively maintained, we recommend considering alternative webmail solutions.

Timeline

Date	Action

sclosure policy

2022-05-03	We inform the vendor that the 90-day disclosure deadline has passed

Summary

In this blog post, we described a vulnerability that allows an attacker to take over a Horde webmail instance simply by sending an email to a victim and having the victim read the email.

The vulnerability occurs in PHP code, which is typically using dynamic types. In this case, a security-sensitive branch was entered if a user-controlled variable was of the type array. We highly discourage developers from making security decisions based on the type of a variable, as it is often easy to miss language-specific quirks.

Related Blog Posts

- RainLoop Webmail - Emails at Risk due to Code Flaw
- Horde Webmail 5.2.22 - Account Takeover via Email
- Zimbra 8.8.15 - Webmail Compromise via Email

SIMON SCANNELL
Vulnerability Researcher