Talos Vulnerability Report

TALOS-2020-1056

Zoom Client Application Chat Code Snippet Remote Code Execution Vulnerability

JUNE 3, 2020

CVE NUMBER

CVE-2020-6110

Summary

An exploitable partial path traversal vulnerability exists in the way Zoom Client version 4.6.10 processes messages including shared code snippets. A specially crafted chat message can cause an arbitrary binary planting which could be abused to achieve arbitrary code execution. An attacker needs to send a specially crafted message to a target user or a group to trigger this vulnerability. For the most severe effect, target user interaction is required.

Tested Versions

Zoom Client Application 4.6.10 Zoom Client Application 4.6.11

Product URLs

https://zoom.us

CVSSv3 Score

8.0 - CVSS:3.0/AV:N/AC:H/PR:L/UI:R/S:C/C:H/I:H/A:H

CWE

CWE-22 - Improper Limitation of a Pathname to a Restricted Directory (‘Path Traversal’)

Details

Zoom is a video conferencing solution that offers a myriad of features. One of the services offered is chat with users contacts. Official client applications exist for Windows, macOS and Linux systems.

Zoom’s chat functionality is built on top of XMPP standard with additional extensions to support rich user experience. One of those extensions supports a feature of including source code snippets that have full syntax highlighting support. The feature to send code snippets requires installation of an additional plugin, but receiving them does not. This feature is implemented as en extension of file sharing support.

In essence, code snippets are shared by generating a special zip archive that contains several supporting files (for untitled code snippet of plain text):

```
Untitled.html
Untitled.properties
Untitled.rtf
Untitled.tx*
```

The final one contains the source itself, the rich text file provides syntax highlighting and the properties file describes the package.

When one user shares a code snippet with another, this zip file is created and uploaded to Zoom’s storage server via `/zoomfile/upload` request to `file.zoom.us`. In return, the Zoom client gets a file object ID and then sends an XMPP message to the intended recipient. The XMPP message looks something like:

```
<message from="source_xmpp_username@xmpp.zoom.us/ZoomChat_pc" to="destination_xmpp_username@xmpp.zoom.us" id="{170029-35B3-4748-9CB0-
42E42FF20DE5}" type="chat">
    <thread>gloox{THREADID}</thread>
    <active xmlns="http://jabber.org/protocol/chatstates"/>
    <sns>
        <format>%1$@ sent you a code snippet</format>
        <args>
            <arg>S E</arg>
        </args>
    </sns>
    <zmext expire_t="1650165620000" t="1587093620200">
    <obj k="key" id="__object_id" s="166" nm="Untitled1.zip" f="14" st="0"/>
    <from n="S E" e="emailaddress" res="ZoomChat_pc"/>
    <to/>
    <visible>true</visible>
    <msg_feature>1024</msg_feature>
    </zmext>
    <body>S E has sent you a code snippet</body>
</message>
```

The object ID attribute uniquely identifies the file that contains the snipped description. When an XMPP client receives the above message, it will proceed to fetch the specified file from Zoom’s data store and will save it to disk with a unique file name. On Windows clients, these files are stored in `%APPDATA%\Roaming\Zoom\data\xmpp_user\CodeSnippet\<random uid dir>`. The same is the case with regular file sharing through Zoom. However, in case of a shared code snippet, Zoom will proceed to automatically unpack the downloaded zip file in order to preview and display the snippet. The core of this vulnerability is that Zoom’s zip file extraction feature does not perform validation of the contents of the zip file before extracting it.

This allows a potential attacker without user interaction to plant arbitrary binaries on target's computer via automatically extracted zip files. Additionally, a partial path traversal issue allows the specially crafted zip file to write files outside the intended randomly generated directory. For example, a file inside a zip archive with a file path of "..\test\another\test.exe" would actually be extracted to `%APPDATA%\Roaming\Zoom\data\xmpp_user\CodeSnippet\test\another\text.exe` instead of being contained inside a directory with random UID. This in itself could potentially be abused in leveraging another vulnerability.

In addition, a quirk of how Zoom handles shared files allows this vulnerability to be taken further with target user interaction. When a regular file is shared with a Zoom client, they need to click on the file and choose a destination to save it before accessing it. Since the Zoom client keeps track of downloaded files, combining this fact with the above described issue can lead to arbitrary file writes to arbitrary paths. In this scenario, the attacker would first share a malicious zip file with the target user with a benign filename, like "interesting_image.jpeg". The target would presumably click and save the file somewhere (on their Desktop for example). The user won't be able to open the file as either a zip or a jpeg directly. Zoom client, on the other hand, keeps track that this particular unique file was saved at the specified path. Then, an attacker sends a code snippet sharing message to the target but specifies the same file id and details in the `obj` tag. Zoom client application will see that the file has already been downloaded and will proceed to unzip it, disregarding the `.jpeg` extension. By abusing the partial directory traversal, malicious zip file could extract files to `c\Users\<username>\` and any children directories.

In this scenario, the attacker uploads a single file to `file.zoom.us` server called `interesting_image.jpeg`. Attacker then sends a message like the following:

```
<message from="source_xmpp_username@xmpp.zoom.us/ZoomChat_pc" to="destination_xmpp_username@xmpp.zoom.us" id="{170029-35B3-4748-9CB0-
42E42FF20DE5}" type="chat">
    <thread>gloox{THREADID}</thread>
    <active xmlns="http://jabber.org/protocol/chatstates"/>
    <sns>
        <format>%1$@ sent you a file</format>
        <args>
            <arg>S E</arg>
        </args>
    </sns>
    <zmext expire_t="1650165620000" t="1587093620200">
    <obj k="key" id="__object_id_of_interesting_image.jpeg" s="166" nm="interesting_image.jpeg" f="5" st="0"/>
    <from n="S E" e="emailaddress" res="ZoomChat_pc"/>
    <to/>
    <visible>true</visible>
    <msg_feature>8</msg_feature>
    </zmext>
    <body>S E has sent you a code snippet</body>
</message>
```

The client saves the file and then the attacker sends another message with almost the same content:

```
<message from="source_xmpp_username@xmpp.zoom.us/ZoomChat_pc" to="destination_xmpp_username@xmpp.zoom.us" id="{170029-35B3-4748-9CB0-
42E42FF20DE5}" type="chat">
    <thread>gloox{THREADID}</thread>
    <active xmlns="http://jabber.org/protocol/chatstates"/>
    <sns>
        <format>%1$@ sent you a file</format>
        <args>
            <arg>S E</arg>
        </args>
    </sns>
    <zmext expire_t="1650165620000" t="1587093620200">
    <obj k="key" id="__object_id_of_interesting_image.jpeg" s="166" nm="interesting_image.jpeg" f="14" st="0"/>
    <from n="S E" e="emailaddress" res="ZoomChat_pc"/>
    <to/>
    <visible>true</visible>
    <msg_feature>1024</msg_feature>
    </zmext>
    <body>S E has sent you a code snippet</body>
</message>
```

The change in the above message is in `f` attribute of `obj` tag. It specifies `14` indicating a code snippet feature. Likewise, the `msg_feature` is adjusted to `1024` as is observed for shared code snippets. The file ID and name inside the `obj` tag are unchanged causing the Zoom client not to re-download a file into it's intended "CodeSnippets" directory, but to use the previously saved path.

It should be noted that even if the target user deletes the saved file upon realizing it is bogus, Zoom client will re-download it but still honor the original save path when the final message is received. Also, malicious zip file can contain multiple copies of malicious files with directory traversing paths which can be used to accommodate for arbitrary places where a target user might save the file.

In summary, this vulnerability can be abused in two above outlined scenarios. First, without user interaction, it can be abused to plant arbitrary binaries on target system albeit at a constrained path potentially used in exploiting another vulnerability. Secondly with user interaction, plant binaries at almost arbitrary paths and can potentially overwrite important files and lead to arbitrary code execution.

Timeline

2020-04-16 - Vendor Disclosure
2020-04-21 - Sent to vendor via a different channel at their request
2020-04-21 - Vendor acknowledged
2020-04-29 - Vendor requested additional information re: OS system tested on
2020-04-30 - Talos retests and issues revised advisory
2020-05-13 - Talos follow up
2020-05-26 - Talos 2nd follow up
2020-05-27 - Vendor says issue was not reproducible in 5.0.1 released on 2020-04-30
2020-05-28 - Talos confirms the issue was fixed in 4.6.12 and confirms 4.6.10 and 4.6.11 (and likely prior) were vulnerable
2020-06-03 - Public Release

CREDIT

Discovered by a member of Cisco Talos.