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

Pentest-Report Obsidian Clients & UI 09.2024

Cure53, Dr.-Ing. M. Heiderich, M. Pedhapati, C. Luders, Dr. D. Bleichenbacher, Dr. N. Kobeissi

Index

Introduction

"Obsidian is the private and flexible writing app that adapts to the way you think. Obsidian stores notes on your device, so you can access them quickly, even offline. No one else can read them, not even us. Obsidian uses open, non-proprietary files, so you're never locked in, and can preserve your data for the long term."

From https://obsidian.md/

This report describes the results of a security assessment of the Obsidian Client software, with an explicit focus on the client. The project, which included a penetration test and a dedicated source code audit, was conducted by Cure53 in September 2024.

The audit, registered as *DYL-03*, was requested by Dynalist Inc. in August 2024. It should be clarified that this is not the first security-centered cooperation between Cure53 and Obsidian. In fact, the Obsidian client itself was analyzed by Cure53 just last year, in November 2023, during the project designated as *DYL-01*. This means that previous pentests and source code audits against the Obsidian client, which revealed several vulnerabilities, could inform this second iteration of examinations targeting the same scope.

In terms of the exact timeline and specific resources allocated to *DYL-03*, Cure53 has completed the research in CW38 In order to achieve the expected coverage for this task, a total of four days were invested. In addition, it should be noted that a team consisting of five senior testers was formed and assigned to the preparation, execution, documentation, and delivery of this project. Given the nature of the tasks envisioned for *DYL-03*, the assessment was contained to a single work package (WP):

- **WP1**: Crystal-box pentests & code audits against Obsidian clients & UI

As the title of the WP indicates, the so-called crystal-box methodology was used. Cure53 was provided with invites to the Obsidian GitHub repository, links to the relevant documentation, as well as all further means of access required to complete the tests.

The project was completed without any major issues. To facilitate a smooth transition into the testing phase, all preparations were completed in CW37. Throughout the engagement, communications were conducted through a private, dedicated, and shared Discord channel. Stakeholders - including Cure53 testers and internal staff from Obsidian - were able to participate in discussions in this space.

Cure53 did not need to ask many questions, and the quality of all project-related interactions was consistently excellent. The continuous exchange contributed positively to the overall results of this project. Significant roadblocks were avoided thanks to clear and careful preparation of the scope. Cure53 provided frequent status updates on the examination and emerging findings, but live reporting was not specifically requested for *DYL-03*.

The Cure53 team achieved very good coverage of the WP1 objectives. Of the eight security-related discoveries, six were classified as security vulnerabilities and two were classified as general weaknesses with lower exploitation potential. It should be noted that the list of findings is not extensive, yet the overall results are still mixed. In particular, based on the observed issues, it can be argued that the security standing of the Obsidian client component has improved since the previous pentest conducted by Cure53.

This is evident from the fact that only one very serious vulnerability was spotted. The problem, identified in the handling of *bookmarks,* has led to UXSS (see DYL-03-007). Positive developments can also be seen in the fact that the findings from the last project, as well as CVEs, have mostly been resolved correctly. The only exception without successful mitigation encompasses CVE-2022-36450, which still appears to be exploitable but can be seen as less impactful at present. Last but not least, many additional hardening suggestions are presented in this report. These are advised as a way to tackle various anti-patterns such as URL spoofing and flawed *deep-links*. Especially the browser WebView plugin should be further hardened to address security risks.

The following sections first describe the scope and key test parameters, as well as how the work packages were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories. The vulnerabilities assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice is provided where applicable. The report ends with general conclusions relevant to this September 2024 project. Based on the test team's observations and the evidence collected, Cure53 elaborates on the overall impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Obsidian clients software complex, more specifically the client component.

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

Scope

- **Penetration tests & source code audits against Obsidian clients**
    - **WP1:** Crystal-box pentests & code audits against Obsidian clients & UI
        - **Obsidian sources:**
            - **Branch:**
                - obsidian-master/
            - **Commit ID:**
                - 220b580d4fd4ab13250703a3efd36310d1b7f0f2
        - **Obsidian static sources:**
            - **Branch:**
                - obsidian-static-master/
            - **Commit ID:**
                - fbf3d1ab4fb01047e36e0b571a5f020268137650
            - **Documentation:**
                - *Local_dev_environment_guide.md*
    - **Test-supporting material was shared with Cure53**
    - **All relevant sources were shared with Cure53**

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., *DYL-03-001*) to facilitate any follow-up correspondence in the future.

DYL-03-001 WP1: Flawed remediation of *CVE-2022-36450* (*Low*)

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

While analyzing the patches applied to resolve prior CVEs, the remediation for *CVE-2022-36450*[1] was found to be flawed. The implemented *file://* scheme check assumes that the string value of *URL(url).protocol*[2] does not contain a colon, which is incorrect. Such assumption leads the application to improperly accept a *file://* URL as safe.

From this perspective, the project remains vulnerable to the same attack. On the plus side, the application's latest version prompts the user for confirmation before opening any *file://* URL, which minimizes the risk associated with this attack.

Affected file:
src/app/workspace/workspace.ts

Affected code:
```
let handleXCallback = (params: ObsidianProtocolData, file: TFile): boolean
=> {
      if (params.hasOwnProperty('x-success')) {
            let url = params['x-success'];
[...]
            let parsed = new URL(url);

            // Don't allow file protocols to be opened from x-callback-url
            if (parsed.protocol.toLowerCase() === 'file') return true;

            window.open(url);
```

To mitigate this issue, Cure53 recommends adding a colon to the *file* string (*file:*) in the *if* statement. This approach will properly remove the risk potential associated with such cases.

[1] https://nvd.nist.gov/vuln/detail/CVE-2022-36450
[2] https://developer.mozilla.org/en-US/docs/Web/API/URL/protocol

DYL-03-002 WP1: *Deeplink* opens arbitrary URLs and leaks filenames *(Low)*

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

Cure53 discovered that the *Deeplink* action in *obsidian://hook-get-address*[3] can be paired with the *?x-success=* parameter to open arbitrary website URLs, as well as to leak the currently opened file's name and full local path. Knowing such sensitive data can be weaponized to force the application to edit and overwrite the file's content.

For the attack to succeed, the victim is required to open a malicious website and accept the browser's prompt to open Obsidian two times. Consequently, the impact score of this flaw is set to *Low.*

PoC code:
```
<script>
document.addEventListener('DOMContentLoaded', () => {
       const p = new URLSearchParams(location.search);
       const s = window.location;
       const n = p.get('name');
       document.body.innerHTML = `<a href="obsidian://hook-get-address?x-
success=${s}">Hook</a>`;
       if (n) {
              const url = `obsidian://new?
content=pwned&silent=1&overwrite=1&name=${encodeURIComponent(n)}`;
              window.location.href = url;
       }
});
</script>
```

Ideally, no data from the user's vault should be shared without explicit permission. This needs to be considered against the necessity to properly display the targeted URL that is being opened. If possible, the *x-callback-url* parameters functionality should be reviewed for more robust transparency and authorization from the user's perspective.

[3] https://help.obsidian.md/Extending+Obsidian/Obsidian+URI#Integrate+with+Hook

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

DYL-03-003 WP1: DoS caused by missing limits in window-opening (*Medium*)

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

During the security assessment, the observation was made that the internal WebView implementation lacks any check or user activation before making it possible to open new windows. Such behavior can be abused by an attacker to create a large number of tabs, causing a Denial-of-Service of the entire application. The state remains even after restarting the application, as it will try to load the malicious website again.

Permitting multiple windows to be open also partially defeats the purpose of the *Cross-Origin-Opener-Policy* security header. This is because several *XSLeaks*[4] attack strategies require continuous window control.

PoC code:
```
<script>
	while(true){window.open("https://cure53.de")}
</script>
```

To mitigate this issue, Cure53 advises allowing only one pop-up to be opened per user interaction, such as a mouse click or pressing a key. This represents an industry standard adopted by most browsers.

DYL-03-004 WP1: URL spoofing via filtered ports (*Medium*)

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

Investigating the security properties of the internal WebView[5] implementation led to the discovery that it is possible to spoof the URL displayed inside the address bar. This can specifically be achieved by navigating to a filtered port.

Behaviors like this occur due to the address bar being updated before the navigation finishes loading. Thus, the previously rendered page is being retained until the response is received from the URL associated with navigation. The timeout necessary to display a proper error message is considerable and not reliable.

PoC code:
```
<h1>Spoofed Page</h1>
<script>
location = "https://obsidian.md:100/";
</script>
```

[4] https://xsleaks.dev/
[5] https://www.electronjs.org/docs/latest/api/webview-tag

To mitigate this issue, Cure53 recommends either updating the address bar only after receiving the response or immediately replacing the previously displayed page with a blank page following navigation.

DYL-03-005 WP1: URL spoofing via redirect to invalid protocols *(Medium)*

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

In the course of the assessment of the internal WebView, another URL spoofing exploit was found. In this case, URLs with invalid protocols are displayed in the address bar before completion of the loading process. This is effectively happening when the previously rendered page is displayed. The invalid protocol scheme is fully shown, but it may be unnoticeable to the user, as it is possible to include punctuation such as *https.://*.

PoC code:
```
<h1>Spoofed Page</h1>
<script>
location = "https.://obsidian.md/";
</script>
```

To mitigate this issue, Cure53 recommends changing the order of visual updates in the application. To be precise, requests should be resolved prior to any altering of the address bar. In this particular case, WebView should promptly display an error page, denoting that the URL is invalid.

DYL-03-007 WP1: UXSS via *bookmarks* accepting JavaScript URI *(Critical)*

***Fix note**: The issue was fixed by the Obsidian team after the audit, and the patch was successfully verified by the Cure53 team.*

While investigating the internal WebView, it was noticed that the user can add any web page URL as a so-called *bookmark*. Once this page is clicked on in the *Bookmark* menu, the URL will be loaded automatically in the internal WebView.

Additionally, the application does not let users input custom b*ookmark* URLs. Bookmarks can only be created for web pages currently loaded in the WebView component. Due to the WebView's HTTP/HTTPS restriction, the addition of *bookmarks* with JavaScript URIs from the application's UI is prevented.

However, Obsidian stores *bookmark* details in a JSON file (*.obsidian/bookmarks.json*) within the vault directory. This file is blindly trusted, allowing adversaries to modify a *bookmark*'s URL to a malicious JavaScript URI from here. Clicking on this modified *bookmark* could load and potentially run arbitrary JavaScript code within the WebView component.

This vulnerability can be exploited easily by malicious actors who gain access to a shared vault through Obsidian. In a shared remote vault, any changes made by any party will be reflected for everyone. This can be exploited in both scenarios:

1. A victim sharing a remote vault with an attacker;
2. An attacker sharing a vault with the victim.

What is more, attackers can trick victims into opening untrusted folders. In both cases, attackers can modify *bookmark* URLs and turn them malicious, potentially facilitating execution of harmful code within the WebView.

If the victim, for example, opens *https://cure53.de* in the WebView mode and then clicks on a malicious *bookmark*, the JavaScript code will be executed within the context of *cure53.de*'s origin. This means that the attacker could perform any actions within that website, as long as the given item was allowed to the victim.

Similarly, this vulnerability could be exploited with other origins, including the *file://* origin. Besides manipulating data on a website, attackers could potentially steal local files.

This attack could be further simplified to include just two clicks, one to load the target website from which the attacker wants to leak data, and the second one to execute arbitrary JavaScript code within that website's context.

Affected file:
src/app/plugins/bookmarks/bookmarks.ts

Affected code:
```
else if (item.type === 'url') {
       leaf.setViewState({
              type: BROWSER_VIEW_TYPE,
              active: true,
              state: {
                     url: item.url,
                     navigate: true,
              }
       });
}
```

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

Steps to reproduce:

1. Create a new vault and enable *Sync & WebView* features.
2. Open any URL in the WebView , click on the *Add Bookmark* option and save it.
3. Open the following *.obsidian/bookmarks.json* file file in text editor from the vault directory.
4. Replace the file content with the following JSON:

   Replaced content:
   ```
   {"items":[{"type":"url","ctime":1727061371528,"url":"https://
   www.google.com/","title":"Click me to load google.com"},
   {"type":"url","ctime":1727061371528,"url":"javascript:alert('UXSS
   window.origin: '+ window.origin +'\n\n Contents: ' +
   document.body.innerHTML)","title":"Click me to alert window.origin
   and also contents"},
   {"type":"url","ctime":1727061371528,"url":"file:///C:/Windows/
   System32/Drivers/etc/hosts","title":"Click me to load
   file:///etc/hosts"}]}
   ```

5. In the Obsidian app, click on the first *bookmark.* This will load *https://google.com*.
6. Click on the second *bookmark*. This will pop the contents. Retrying the sequence with the third *bookmark* loads a local file.

Cure53 advises utilizing an URL parser API to validate the URL scheme of the *item.url* property. It needs to be ensured that only HTTP/HTTPS schemas are permitted in the context of opening items from the *Bookmark* bar.

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, while a vulnerability is present, an exploit may not always be possible.

DYL-03-006 False Positive: Outdated & vulnerable dependencies in Obsidian static *(Info)*

Note: The issue is a false positive, after discussing with the client the vulnerable modules identified by npm audit are all dev dependencies only used in the build process and not deployed/exposed to the public.

Several software packages observed during the assessment were seen as reliant on outdated versions of components. As such, the complex was unnecessarily vulnerable to a host of security risks.

The software packages listed below were identified as out-of-date and potentially insecure. Notably, the version information provided is based on data collected at the time of testing. Whether these vulnerabilities are exploitable depends entirely on how the relevant functionality is used in the targeted application at present.

Command:

```
/obsidian-static-master> npm audit --omit dev

# npm audit report

body-parser  <1.20.3
Severity: high
body-parser vulnerable to denial of service when url encoding is enabled -
https://github.com/advisories/GHSA-qwcr-r2fm-qrc7
[...]

braces  <3.0.3
Severity: high
Uncontrolled resource consumption in braces -
https://github.com/advisories/GHSA-grv7-fg5c-xmjg
[...]

ejs  <3.1.10
Severity: moderate
ejs lacks certain pollution protection -
https://github.com/advisories/GHSA-ghr5-ch3p-vcr6
[...]
```

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

```
micromatch  <4.0.8
Severity: moderate
Regular Expression Denial of Service (ReDoS) in micromatch -
https://github.com/advisories/GHSA-952p-6rrq-rcjv


path-to-regexp  <=0.1.9 || 4.0.0 - 6.2.2
Severity: high
path-to-regexp outputs backtracking regular expressions -
https://github.com/advisories/GHSA-9wv6-86v2-598j
[...]

postcss  <8.4.31
Severity: moderate
PostCSS line return parsing error - https://github.com/advisories/GHSA-
7fh5-64p2-3v2j
[...]

pug  <=3.0.2
Severity: moderate
Pug allows JavaScript code execution if an application accepts untrusted
input - https://github.com/advisories/GHSA-3965-hpx2-q597
[...]

send  <0.19.0
Severity: moderate
send vulnerable to template injection that can lead to XSS -
https://github.com/advisories/GHSA-m6fv-jmcg-4jfg
[...]


ws  8.0.0 - 8.17.0
Severity: high
ws affected by a DoS when handling a request with many HTTP headers -
https://github.com/advisories/GHSA-3h5v-q93c-6h6q
[...]

11 vulnerabilities (6 moderate, 5 high)
```

Notably, the testing team was unable to comprehensively prove any potential impact during the limited time frame granted for this review. As such, the wider implications remain unknown at this point, and it is recommended that they are subject to internal research at the earliest possible convenience for the in-house team.

Generally speaking, the provision of optimal and robust supply chain security can be quite challenging. Often, an easy or comprehensive solution cannot be offered, while the results and efficacy of the selected protection framework can vary depending on the integrated version of the deployed libraries.

To mitigate the existing issues as effectively as possible, Cure53 recommends updating all affected libraries and establishing a policy to ensure that libraries remain up-to-date moving forward. This will ensure that the premise can benefit from patches rolled out for previously detected weaknesses across a variety of different solutions. For this purpose, NPM offers a functionality entitled *npm audit fix.* Note, however, that the degree of protection may fluctuate; up-to-date retention will typically become increasingly difficult to achieve as a greater number of third-party libraries are deployed.

DYL-03-008 WP1: Markdown permits *file://*-protocol (*Info*)

Note: After discussing with the client, the issue was considered a feature since users are making the choice after displaying the warning.

It was observed that the markdown feature allows the use of the *file://*-protocol. When this is clicked on, a warning message is displayed. However, if the user accepts it, the file at the specified path will be opened using *shell.openPath.*

Although a warning message is displayed, it cannot be excluded that a user disregards it and proceeds. In such scenarios, a compromise would be the consequence. Since the *shell.openPath* API in Electron can execute binaries, this signals potential for RCE.

To fully mitigate this risk, it is strongly recommended to block the possibility to open any files through the app, regardless of warnings.

Conclusions

As signaled in the *Introduction,* the security posture of the Obsidian clients and UI components appears to be improving. Comparing the results of *DYL-01* and *DYL-03* projects, Cure53 can assert proper dedication to fixes and hardening resolutions, which bodes well for the future security standing of Obsidian, as long as this dedication can be maintained at Obsidian. While eight issues were still confirmed as negatively affecting the scope, only one *Critical* flaw was documented during this September 2024 investigation.

Detailing the process and findings, the assessment started with a review of the patches applied for prior CVEs. It was found that the remediation for *CVE-2022-36450* has been done in a flawed manner (see DYL-03-001). Other CVEs were also inspected but no bypass or side-effects were noticed. The application does not have a wide history of vulnerabilities.

Static analysis and security source code review methods were applied to properly inspect the application's registered *Deeplinks* as such functionality is a common attack surface for thick clients. The *hook-get-address* presented a risk to the application, as it can be used to leak the currently opened file's name and full path. Such sensitive information can be abused to overwrite files, issue DYL-03-002. The *x-callback-url* parameters are a point of failure and should be reviewed, with Obsidian aiming for a more robust and transparent implementation.

The new browser plugin implementation was carefully analyzed for common security pitfalls of the WebViews components, with the focus on its Electron interactions. The WebView tag configuration was inspected for security misconfigurations. The implementation neither utilizes any custom and potentially vulnerable *webpreferences* configuration, nor enables unsafe and dangerous attributes like *nodeintegration* or *disablewebsecurity*.

Multiple vulnerabilities related to the browser plugin were found, mainly due to the lack of common security hardening protections of full-fledged browsers. For example, a DoS vulnerability was caused by the WebView's lack of limits for the number of permitted windows opened by a website (DYL-03-003). This entailed loading multiple tabs and crashing the application.

Another common browser-related vulnerability is the ability to spoof the URL address bar and trick the user into thinking another website is loaded, even though a malicious page is being displayed in reality. Two vulnerabilities of such type were found; they arise from the erroneous parsing of the loaded URLs and the determinations on how they should be loaded and displayed in the address bar. More details on these flaws can be found in DYL-03-004 and DYL-03-005.

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

Electron-related security anti-patterns were investigated thoroughly in the application's source code. For example, the *executeJavaScript()* function was found many times, so all of the user-controlled objects used in this context were analyzed for potential JavaScript injections. The application properly utilizes *JSON.Stringify* to escape these inputs before executing the hardcoded JavaScript. One minor issue was noticed regarding the file protocol URLs and their handling upon reaching dangerous sinks (see DYL-03-008).

Overall, it is apparent that a more robust security implementation is needed for the *Deeplink* functionality and this realm should be reviewed in-depth. The browser WebView plugin is also rather fragile from a security perspective, lacking common browser hardening improvements. It is hoped that it can be more battle-tested before a proper, full release. Apart from these observations, Cure53 concludes that most of the Electron anti-patterns were successfully avoided in the Obsidian clients and UI project.

Cure53 would like to thank Erica Xu, Steph Ango, Shida Li, and Tony Grosinger from the Dynalist Inc. team for their excellent project coordination, support and assistance, both before and during this assignment.