Fine penetration tests for fine websites

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

# Pentest-Report Passbolt SSO, API & Add-On 02.-03.2023

Cure53, Dr.-Ing. M. Heiderich, M. Rupp, L. Herrera, BSc. B. Walny

## Index

- Introduction
- Scope
- Identified Vulnerabilities
  - PBL-08-001 WP2: Credentials leakage via clickjacking (High)
  - PBL-08-002 WP2: Passphrase retained in memory post-logout (Low)
  - PBL-08-003 WP1: Lack of proper ACL for users endpoint (Low)
  - PBL-08-006 WP1: 2FA status information disclosure via users endpoint (Info)
  - PBL-08-007 WP1: SSO design prompt=none facilitates auth bypass (Medium)
- Miscellaneous Issues
  - PBL-08-004 WP1: Lack of 2FA login code rate limiting (Info)
  - PBL-08-005 WP1: Lack of cross-origin-related HTTP security headers (Info)
  - PBL-08-008 WP2: Lack of explicit CSP on extension manifest (Info)
- Conclusions

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

## Introduction

*"The password manager your team was waiting for. Free, open source, self-hosted, extensible, OpenPGP based."*

From https://www.passbolt.com/

This report documents the scope, coverage, and findings concerning a Cure53 penetration test and source code audit against the Passbolt platform. The testing team placed primary focus on its SSO features, backend API endpoints, browser add-on, and frontend aspects.

The audit was requested by Passbolt S.A. in December 2022 and conducted in late February and early March 2023, specifically in CW09. In order to achieve the expected coverage for this project, a total of ten days were invested. All assessment actions were divided into two distinct work packages (WPs) for fluid execution. These read as follows:

- **WP1**: White-box tests & security assessments against Passbolt SSO features & backend API
- **WP2**: White-box tests & security assessments against Passbolt SSO browser add-on & frontend parts

This audit marks the eighth collaborative engagement between Passbolt S.A. and Cure53, though the Passbolt SSO constitutes a newly-implemented feature and as such has not been included in scope for any prior test iterations.

To facilitate optimum coverage, a testing instance, sources, test-user accounts, pertinent supporting information and documentation, and all other access elements deemed necessary were provided. The selected methodology was white-box, whilst the preparation, delivery, and completion of this security review was handled by four skill matched senior testers from the Cure53 team. Preparation procedures were accomplished in CW08 February 2023 to ensure no blockers would be encountered ahead of testing.

Communications between the Passbolt S.A. and Cure53 teams were enabled by a dedicated, shared Slack channel, for which all relevant staff from both sides were invited to partake. As a result, cross-team discussions were smooth on the whole. The scope received sufficient preparation, any test-related queries were kept to a minimum, and the active assessment phase was trouble-free.

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

The tester team relayed numerous status updates concerning the audit progression and noteworthy findings. Live reporting was offered and implemented via the aforementioned Slack channel.

In relation to the issues discovered, Cure53 identified a total of eight findings following strong coverage against the WP1 and WP2 scope items. Over half of the findings (specifically 5) were categorized as *Identified Vulnerabilities*, whilst all remaining exhibited minor exploitation potential and were consequently assigned to the *Miscellaneous Issues* section. This outcome was warmly received by Cure53, since the total yield of findings is relatively moderate compared with other correlatory audit scopes.

In addition, the testing team positively acknowledged that any *Critical* vulnerabilities were effectively negated by the platform's security resilience. However, one particular issue pertains to the potential leakage of credentials via a clickjacking scenario, as stipulated in ticket PBL-08-001. As a result, this defect's severity impact was upgraded to *High*. Nevertheless, the Passbolt team's swift mitigation of said issue is commendable and attests to its commitment to high security standards for the components in scope.

To summarize, Cure53 is pleased to confirm that the Passbolt platform already exhibits sufficient protection against a plethora of varying attack scenarios. Nevertheless, as corroborated by the volume of tickets raised in this report, a number of hardening opportunities are present that require follow-up actions from the developer team in order to raise the security offering even further.

The scope and test setup, as well as the material available for testing, are further clarified below. Following this, the report enumerates all findings in chronological order, with the detected vulnerabilities detailed initially then proceeded by all general weaknesses. Each finding offers technical analysis, a PoC where necessary, and optimal mitigation or fix advice.

Finally, Cure53 will elaborate on the general impressions gained throughout this test in the conclusion section, which serves to offer a transparent and comprehensive overview of the scope's perceived security posture.

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

## Scope

- **White-box penetration tests against Passbolt SSO, API, & browser add-on**
  - **WP1**: White-box tests & security assessments against Passbolt SSO features & backend API
    - **Test instance:**
      - https://pro.passbolt.dev/
    - **Source code:**
      - https://bitbucket.org/passbolt_pro/passbolt_pro_api/src/release/plugins/PassboltEe/Sso/
  - **WP2**: White-box tests & security assessments against Passbolt SSO browser add-on & frontend aspects
    - **Source code:**
      - https://github.com/passbolt/passbolt_browser_extension/tree/release
    - **Test user-accounts utilized in this assessment:**
      - E: rupp@cure53.de
      - E: maxim@cure53.de
      - E: rupp+user@cure53.de
      - E: herrera@volt.cure53.de
      - E: ben@cure53.de
  - **Test-supporting material was shared with Cure53:**
    - Part I - SSO admin settings & login
    - Part II - Recover process (browser reconfig process)
    - Passbolt styleguide
  - **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

All security and implementation issues identified during testing are provided below. Findings are enumerated in chronological order rather than by their degree of severity and impact. The severity rank is offered in brackets after the title heading for each vulnerability, which precedes a unique identifier (e.g. *PBL-08-001*) to assist with any follow-up reporting.

### PBL-08-001 WP2: Credentials leakage via clickjacking *(High)*

***Fix note**: This finding was mitigated during active testing and a temporary mitigation verification was issued by Cure53.*

The observation was made that all the files from the *webAccessibleResources* folder are externally exposed to web pages via a wildcard in the *web_accessible_resources* manifest property. Files that are listed in this particular property can be navigated to or framed by any arbitrary page.

Following extensive analysis, testing confirmed that two of these files - namely *quickaccess.html* and *passbolt-iframe-page.html* - facilitate credential leakage and other malicious actions via a clickjacking attack. Notably, the SSO feature functions similarly to the scenario whereby users select the *Remember until I log out* option, which is a necessary step to successfully instigate this attack vector.

The credentials can be leaked by a malicious page by framing the *quickaccess.html* page, then creating *username* and *password* input fields. Subsequently, the user will be tricked into performing four random clicks on the attacker's page.

Unbeknownst to the user, they will actually click on *Filters*, then *Items I own*, a random credential, and finally the *Use on this page* option - all of which populates the attacker's form with the victim's credentials.

**Affected file:**
*/passbolt_browser_extension-release/src/all/manifest.json*

**Affected code:**

```
"permissions": [
   "activeTab",
   "clipboardWrite",
   "tabs",
   "storage",
   "unlimitedStorage",
   "*://*/*",
```

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

```
    "alarms"
  ],
  "web_accessible_resources": [
    "webAccessibleResources/*"
  ]
```

**Steps to reproduce:**

1. Log into the Passbolt extension using SSO.
2. Ensure you own at least one credential, then navigate to the following URL: *https://lbherrera.github.io/lab/passbolt-9910038/index.html*.
3. Note that an iframe with opacity will be displayed. Here, click on *Filters*, followed by *Items I own*, your credentials, and finally on the *Use on this page* option.
4. Observe that an alert will be displayed containing the user's credentials.

In a tangible attack scenario, the victim would only need to make four arbitrary clicks on the attacker's webpage and would be oblivious to the fact they are interacting with the extension. The PoC provided does not completely hide the iframe for demonstration purposes and no further effort was required to render the clicks more seamless.

To mitigate this issue, Cure53 recommends removing the affected files from the *webAccessibleResource* property. If this is deemed infeasible due to certain feature requirements, the *frame-ancestors* CSP directive should be specified in each affected file to ensure only safe, allow-listed domains are contained.

### PBL-08-002 WP2: Passphrase retained in memory post-logout *(Low)*

The observation was made that the passphrase of a given account is not cleared from memory immediately following user logout. This means that an attacker with physical access to the victim's computer can retrieve the victim's passphrase, even if the account is logged out. Notably, this attack vector is only possible if the extension popup has not been closed since the account was last logged in and logged out.

Furthermore, the testing team noted that the passphrase is not cleared from memory immediately after login in the event the user has not selected the *Remember until I log out* option.

Nevertheless, the fact that the memory is cleared once the popup is dismissed significantly reduces the severity impact of this issue, with the ticket appropriately downgraded to *Low*.

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

**Steps to reproduce:**

1. Ensure Google Chrome is used and the Passbolt extension is installed.
2. Log in using the passphrase.
3. Click *Logout*.
4. Right-click on the Passbolt extension popup and click *Inspect*.
5. Click the *Memory* tab on the DevTools window that appears.
6. Click *Take snapshot*.
7. Press *CTRL+F* and type the start of your Passbolt passphrase after the snapshot is created.
8. Observe that the password is identifiable in the memory.

To mitigate this issue, Cure53 advises closing the Passbolt extension popup and reopening it after the user logs out to ensure memory is sufficiently cleared.

### PBL-08-003 WP1: Lack of proper ACL for *users* endpoint *(Low)*

The discovery was made that the *users* functionality suffers from a lack of correct access-control rules on the server-side. This can be observed when a malicious *User*-role user attempts to access the affected endpoint by using a third-party resource ID for the *filter[has-access]* parameter. The approach succeeds, even though the item should be restricted and unavailable to users without access for the requested resource. In the current implementation, the endpoint's response discloses a list of members with legitimate resource access to the malicious user. This type of action should clearly be blocked within the presently utilized access model.

The following example demonstrates the erroneous behavior. Note that the included requests to the application utilize the *rupp+user@cure53.de* (*User* role) user-account session, which lacks resource access. Said user can successfully view the list of members or groups that share the resource (ID: *41899b09-aa7a-49a7-995c-b8f2fe371f25*), even if this action is not permitted by the role.

**Request:**

```
GET /users.json?api-version=v2&filter%5Bhas-access%5D=41899b09-aa7a-49a7-995c-
b8f2fe371f25 HTTP/1.1
Host: pro.passbolt.dev
Cookie: passbolt_session=ts935mnlfl7j9po8p6nq4c4hs8
```

**Response body:**

```
[...]
"body": [
     {
         "id": "4a13c02a-d06e-4814-9a7d-a03c21ff6c77",
         "role_id": "0d51c3a8-5e67-5e3d-882f-e1868966d817",
```

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

```
            "username": "herrera@volt.cure53.de",
            "active": true,
            "deleted": false,
            "created": "2023-02-27T11:04:25+00:00",
            "modified": "2023-02-27T19:58:47+00:00",
            "groups_users": [],
[...]
```

Given the limited exploitability of this issue for instigating a tangible attack, the severity rating was appropriately downgraded to *Low*.

Generally speaking, correct access control should never solely rely on the assumption that a user is unaware of the methods by which they can reach various endpoints.

To mitigate this issue, Cure53 advises implementing adequate authorization checks for the affected application endpoint. The developer team's amended approach should determine whether a user with the requested session is permitted to obtain an endpoint's contents via the specified requests. If this is not the case, the backend should return the associated error code.

### PBL-08-006 WP1: 2FA status information disclosure via *users* endpoint *(Info)*

The observation was made that a user's Two-Factor Authentication (2FA) status is disclosable by a malicious user via the */users/* API endpoint. Nevertheless, Cure53 would like to underline that this status is only displayed to administrators in the application's web interface. Despite the fact that this behavior does not directly facilitate significant risk, user information may be unnecessarily revealed and as such should be addressed. The following example relating to one of the affected endpoints demonstrates the present behavior.

**Request:**

```
GET /users/407cb1be-3ab5-4840-99f2-391131f2bd74.json HTTP/1.1
Host: pro.passbolt.dev
Cookie: passbolt_session=jg8auftkkhlh7ev281tv480b2m;
csrfToken=ea43450116de96ca7909bad68d038f35edbee5ae22c6338fe65b95c41bc508b8f7e5dc
dd346323d901f89f7a42ba5749ae9989788eed163c1bfea9ccd6aa3714
```

**Response body:**

```
[...]
"is_mfa_enabled": false,
[...]
```

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

As one can deduce from the response, the endpoint may return previously-saved information concerning the user's two-factor authentication status. However, since this finding does not leak any pertinent, critical information, the severity marker was appropriately downgraded to *Info*.

In spite of the minor risk potential in this regard, Cure53 hopes that this issue is viewed in context, considering that it will generally assist attackers in their efforts to enumerate alternative methods to conduct further attacks against the target.

To mitigate this issue, Cure53 advises altering the application's logic flow to ensure that the affected endpoint also censors the user's 2FA status for the *User* role.

### PBL-08-007 WP1: SSO design *prompt=none* facilitates auth bypass *(Medium)*

Whilst auditing the Microsoft Single-Sign On (SSO) implementation, the observation was made that the application's enforcement of the SSO *prompt=login* upon login necessitates users to always authenticate with their Microsoft password. Here, the SSO's intention to negate the need to memorize two passwords - i.e. the passphrase for the Passbolt application and an SSO password - and leverage a single password only was deemed susceptible to risk in the event of a local attack.

Whilst remote attackers with XSS capabilities would still be prompted for the password, local attackers could circumvent the authentication altogether by intercepting the SSO login request and replacing the prompt parameter from *login* to *none*. The fact that the Passbolt application's logout function does not invalidate the SSO sessions at the IdP - which enables a local attacker to access all saved passwords without further knowledge - is pertinent for this compromise scenario.

Even security-conscious users that frequently log out of their password managers may remain unaware that the inherent nature of SSO will render them susceptible to risk, particularly in relation to non-invalidated sessions.

To reproduce this vulnerability, simply follow the steps offered below:

**Steps to reproduce:**

1. Sign into the Passbolt application once with SSO, regardless of any *Keep my session* setting on the Microsoft side.
2. Sign out.
3. Intercept the outgoing requests from the browser extension via Burp Suite, for instance.

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

4. Click *Sign in via SSO*, then alter the intercepted *[..]/oauth2/v2.0/authorize/?[..]* request to *login.microsoftonline.com* and the parameter *prompt* from *login* to *none*.
5. Forward all other requests and desist interceptions.
6. Observe that the account is logged in.

Given the local attacker requirement, Cure53 considers this a *Medium* severity vulnerability. However, in a password manager context, this behavior is deemed to incur significantly greater consequences and essentially bypasses its fundamental design guarantee.

To mitigate this issue, Cure53 recommends adopting one of three approaches. Firstly, one could explicitly state that SSO usage facilitates additional risk regarding local attackers, i.e. a warning stipulating that a ‘logout’ does not technically represent a logout in actuality. Secondly, one could disallow usage of *prompt=none* at the IdP side. Finally, the developer team could adopt SSO logout features to invalidate tokens following a Passbolt application logout. The latter solution, however, will likely negatively impact UX, since other services leveraging the SSO may also be affected by this implementation.

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

## Miscellaneous Issues

This section of the report pertains to all noteworthy findings that did not incur exploitation potential but may prove useful for an attacker in their efforts to successfully compromise the framework in the future. In general, the majority of these tickets represent vulnerable code snippets that were considered challenging to call. In summary, whilst a security weakness persists for all findings noted here, an exploit is likely implausible.

### PBL-08-004 WP1: Lack of 2FA login code rate limiting *(Info)*

Testing confirmed that the application has not established any rate-limiting protection from incoming attempts whilst 2FA functionality is enabled, which can be leveraged to instigate brute-force attacks against the mechanism of the required code.

Notably, more than 200 failed attempts were attempted during testing. However, a correct 2FA code was subsequently accepted. Therefore, an attacker that is able to obtain the victim's credentials can continue to input random codes until a successful hit is achieved.

To mitigate this issue, Cure53 advises implementing a rate-limiting strategy for 2FA code requests by restricting access to an account after a certain volume of failed attempts.

### PBL-08-005 WP1: Lack of cross-origin-related HTTP security headers *(Info)*

Testing validated that the Passbolt platform lacks several of the latest Cross-Origin-infoleak-related HTTP security headers in its responses[1]. This behavior does not directly incur a security weakness per se, though may assist attackers in their attempts to exploit other potentially compromisable areas, particularly concerning Spectre attacks[2] and similar. The developer team should incorporate the following headers to ensure comprehensive protection against all 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 - and further prevent data delivery to an attacker-controlled browser-renderer process, as observed for *resourcepolicy.fyi* and *web.dev/fetch-metadata*.
- **Cross-Origin Opener Policy (COOP)** permits the ability to ensure that the application window will not receive unexpected interaction from other websites, thereby facilitating browser isolation for its own processes. This is considered a

[1] https://security.googleblog.com/2020/07/towards-native-security-defenses-for.html
[2] https://meltdownattack.com/

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

crucial process-level measure, particularly for browsers that do not enable full Site Isolation; please refer to *web.dev/coop-coep* for supplementary guidance.

- **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 modern era, applications must enable both COEP and COOP to sufficiently guarantee process-level isolation for highly sensitive applications in Chrome or Firefox, as stipulated via *web.dev/coop-coep.*

Generally speaking, the continued absence of cross-origin security headers is suboptimal and should be addressed, particularly considering the public availability of exploit code and proliferation of attack scenarios such as Spectre.

To mitigate this issue, Cure53 advises integrating the aforementioned headers into every relevant server response. Resources for these headers are available online, which extrapolate optimum header setup implementation[3] plus the many differing consequences of omitting them entirely[4].

### PBL-08-008 WP2: Lack of explicit CSP on extension manifest *(Info)*

The observation was made that the Passbolt extension currently does not explicitly set the *content_security_policy* manifest key on its *manifest.json* file and as such relies on the default policy applied. This security feature serves to provide auxiliary defense-in-depth, since integration permits policy definition for specific HTML tags, such as script elements, which include the origin a resource can be loaded from and similar. Essentially, this feature's raison d'etre is to ensure that abusive HTML injection is either completely deterred or rendered highly difficult to achieve.

To mitigate this issue, Cure53 advises integrating additional directives to the CSP ruleset, such as *base-uri*, *form-action*, and *frame-ancestors*. Despite the fact that the default configuration is adequately strict and generally considered secure, these directives will certainly help to improve the extension's security posture and should therefore be implemented.

[3] https://scotthelme.co.uk/coop-and-coep/
[4] https://web.dev/coop-coep/

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

## Conclusions

The impressions gained during this report - which details and extrapolates on all findings identified during the CW9 testing against the Passbolt, browser extension, surroundings, and new connected SSO feature by the Cure53 team - will now be discussed at length. In summation, all components tested favorably following the completion of this audit, though plenty of opportunities for security hardening were observed.

Notably, this assessment was preceded by a number of previous engagements against the Passbolt scope by Cure53. Similarly to PBL-02 and PBL-03, the testing team noted a manageable and fairly minimal yield of findings. Considering that rigorous evaluations and an ample time frame were required before unearthing any issues during this assessment attests to the positive impression gained. All risk scenarios are evidently meticulously monitored and controlled by the in-house team, thereby facilitating robust security stability for the Passbolt framework.

One can confirm that the focus applications have proven robust against the multitude of attack scenarios instigated from a server- and client-side perspective. The ten-day allocation for this examination yielded a total of eight findings, which is a praiseworthy result for the Passbolt team. The volume and severity markers attached to the findings is moderate for a scope of this magnitude. The absence of any major issues - with no *Critical*-assigned vulnerability in particular - underlines the Passbolt complex's security strength. Even so, the identified flaws represent a golden opportunity to integrate additional safeguard measures. The following paragraphs extrapolate the resulting coverage and findings.

Firstly, Cure53 would like to comment on the project's objectives and tasks initiated in order to achieve them. The testing team's primary focus constituted a review of the backend and frontend; the source codes of the components were also subjected to particular scrutiny. The basic premise of this Cure53 investigation was to determine whether the existing functionality of the endpoints and their environment can be deemed suitably safeguarded to prevent attacks by malicious users seeking to damage the Passbolt complex. The testing team also strived to identify typical modern-application problems and issues associated with various types of injection attacks, which could compromise the application's server or client areas.

Additionally, Cure53 sought to determine the presence of any potential access matrix implementation flaws and leakage of sensitive information via the deployment of application endpoints, with a host of advanced techniques implemented in this respect. Similar approaches were initiated to verify any logic weaknesses that may blight the complex.

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

Concerning the findings specifically, Cure53 will first comment on the most significant findings. The implemented authentication and authorization mechanisms were assessed for common bugs, vulnerabilities, and misconfigurations. Testing was initiated to determine whether the login process was sufficiently protected against a variety of attack scenarios, including authentication bypass and user-session leakage. Positively, no severe findings were noted in this instance, since the session implementation withheld the testing team's numerous manipulation and cracking attempts. However, a design flaw was noted in reference to a potential local attack, which leverages standard SSO features to bypass Passbolt application authentication, as a result of pre-established SSO sessions. This SSO-related vulnerability is considered challenging to resolve; the developer team could simply accept the associated risk or evaluate the optimal method by which to force the logout function to invalidate not only sessions on the Passbolt side but also at the IdP, which would therefore block subsequent login attempts without the passphrase.

The testing team deemed varying types of client-side injection instances pertinent for examination. To help facilitate this, Cure53 investigated the JavaScript code and application functionality for any DOM-based XSS and similar input-manipulation or client-side issues. The HTML handling was also scrutinized in relation to client-side attacks, though these efforts were not particularly fruitful. All outputs were correctly encoded or sanitized before displayed, therefore rendering this attack vector ineffective. In general, the application's evident stability in this regard was positively acknowledged - a rare occurrence for security reviews of this nature. The Passbolt team deserves every plaudit for its dedication to providing airtight security for its products.

Since the API constitutes a key application component, a host of testing techniques were applied against it in an attempt to uncover any typical, commonly-experienced attack vectors. The API endpoints and connected functionality - as well as the body handling at the API endpoints - were subjected to stringent assessment. Since the API endpoints handle JSON bodies, the platform's susceptibility to risk may expand in the event input is insufficiently handled via type confusions, deserialization issues, or flaws related to mass assignment. Given the intrinsic nature of JSON, type confusions may become problematic when a boolean is supplied rather than a string, for instance. Furthermore, the code was assessed for any potentially dangerous calls, such as execution calls, which may otherwise incur RCE or similar injections. Again, these efforts yielded a lack of noteworthy findings.

The testing team faced a number of error messages during the active review phase, which indicates that untrusted user input is carefully validated and sanitized. This helps to reduce any potential attack surface on the whole, with only controlled, predictable, and expected set of user-input instances permitted. Subsequently, the vast majority of

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

attack and injection scenarios remain implausible. As a result, the testing team was unable to locate any untrusted user-input misuse at API endpoints. To conclude here, the backend was perceived to offer an adequate security foundation, particularly in light of the fact that significant risk scenarios have evidently been considered and nullified by the Passbolt team.

Cure53 would also like to underline the inability to uncover any serious issues associated with the access-control matrix, despite intensive and dedicated searches for compromisable pathways. As a general rule, multi-user platforms must always ensure that accounts are only able to read or modify the data to which they have been specifically granted access. The Cure53 team noted that endpoints clearly determine user input and verify whether certain access is available for the user prior to the final acceptance of such input. Henceforth, these approaches could not identify any significantly risk-laden findings. In spite of this, two flaws were uncovered here: one revealing a user's 2FA status, and the other (albeit minor) pertaining to the *users* endpoint. Further guidance on these is offered in tickets PBL-08-003 and PBL-08-006.

Concerning the Passbolt browser extension work package, the frontend components were subjected to a code review and deep-dive analysis to determine any potential for client-side vulnerabilities, including those related to *postMessage* issues, prototype pollution, and DOM XSS sinks. The front end (Styleguide) primarily utilizes the ReactJS library, which leverages a battle-tested escaping mechanism that prevents XSS issues by default. No usage of *dangerouslySetInnerHTML* was observed, which was deemed an astute omission considering its tendency to incur security vulnerabilities.

Elsewhere, the extension's manifest file was extensively evaluated. Here, several files were found to be exposed to the internet via the *web_accessible_resources* property. Supplementary assessments were conducted following this discovery, which led to the detection of a *High*-rated severity issue that facilitated credentials leakage via a clickjacking attack, as documented in ticket PBL-08-001. In addition, the manifest lacks a strict CSP; naturally, Cure53 strongly advises adhering to the advice offered to minimize any potential exploitation of XSS issues (see PBL-08-008).

The testing team also investigated any issues that could incur passphrase leakage, both locally on the user's browser or by the backend. Here, one minor issue was documented whereby a user's passphrase is readable by an attacker with physical access to the user's computer under specific circumstances. Notably, this attack scenario persists even after the user has logged out from the extension (see PBL-08-002).

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

Lastly, the testing team allocated substantial resources toward analyzing pertinent extension code from the new SSO feature. Fortunately, no specific or related issues were consequently encountered, which again corroborates the platform's robustness and general security offering.

The extension offers minimal attack surface, primarily due to the established port-based communication mechanism, which serves to restrict a host of attack vectors. The testers also attempted to construct a malicious SSO and leverage its Login URL as a vector upon which to execute JavaScript, as well as determine whether one could load malicious web pages in the extension's popup. However, the extension's robust URL validation imposes stringent restrictions on the permissible URLs, attesting to the lack of notable vulnerabilities in this regard.

In summation, the assessed complex offers reasonable stability and was positively received by the Cure53 team. Despite the fact that this report stipulates a number of weaknesses and best practice recommendations, most incur limited severity potential or were simply deemed miscellaneous in nature. Cure53 is keen to stress that none of the detected issues allow an attacker to retrieve direct access on the server-side, which in turn means that any sensitive data considered vital to either the platform or its users is unequivocally protected. Furthermore, all identified issues are considered straightforward to resolve and do not constitute fundamental design weaknesses that require complex alterations to administer.

The code base withstood intense testing scrutiny for the most part and was observably effective in minimizing the attack surface. Both the minimal volume and severity impact of the detected issues provide irrefutable evidence that the in-house team has integrated a number of precautions to secure the complex. One can certainly argue that the development team is not only aware of routine security errors but has already implemented sound security initiatives to prevent them.

All in all, the complete negation of any issues connected with injection attacks - which ensure that the platform's server-side remains insusceptible to compromise - serves as one of the primary indicators of security strength. In addition, despite extensive investigations and widespread coverage from the Cure53 testers, the lack of noteworthy API-related findings contributes to the satisfactory outcome achieved following the finalization of this audit. The triumvirate defensive measure of escaping, encoding, and filtering has been implemented to an exemplary standard. Nevertheless, a handful of improvements on the logic and browser extension-side should be actioned at the earliest possible convenience to enhance the complex's security posture.

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

To summarize, Cure53 believes that a first-rate security standard can be achieved by addressing and mitigating all findings documented in this report, though the Passbolt team has evidently already constructed an admirable foundation upon which to cultivate future project investments.

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