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 Browser Extensions 04.2021

Cure53, Dr.-Ing. M. Heiderich, Dr. N. Kobeissi, BSc. T.-C. “Filedescriptor” Hong

Index

Introduction

> *“The password manager your team was waiting for. Free, open source, self-hosted, extensible, OpenPGP based.”*
>
> From https://www.passbolt.com/

This report describes the results of a comprehensive security assessment targeting the Passbolt Browser Extensions for Chrome and Firefox. Carried out by Cure53 in April 2021, the project entailed both a penetration test and a dedicated source code audit of the Passbolt extensions in scope.

To give some context, the work was requested by Passbolt SA in early March 2021 and then scheduled for the following month. Note that this is the second review Cure53 conducted for Passbolt, hence the label *PBL-02*. The first cooperation in the security realm was in February 2021 and can be found in the *PBL-01*.

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

The current project was assigned to a team of three senior testers who prepared, executed and documented the examination. The core work was done in early April 2021, namely in CW14, whereas the overall budget stood at eight person-days. To optimally structure the work needed, two work packages (WPs) were delineated as follows:

- **WP1**: Penetration-Tests & Code Audits against Passbolt Chrome Extension
- **WP2**: Penetration-Tests & Code Audits against Passbolt Firefox Extension

Driven by the aim of acquiring breadth and depth of coverage, white-box methods were used. Cure53 was given access to all relevant sources in an uncompressed form, as well as browser extension builds for Chrome and Firefox. All preparations were done in late March 2021, namely in CW13, so as to enable swift start and efficient progress.

Communications during the test were done using the Slack channel established and used for *PBL-01.* Discussions were slim as the scope was well-prepared and clear. No noteworthy roadblocks were encountered during the test. Just as in *PBL-01*, the Passbolt team was extremely helpful and had a positive impact on the overall testing process. Cure53 offered frequent status updates about the test and findings. Live-reporting was not requested.

The Cure53 team managed to get very good coverage over the WP1-2 scope items. Two discoveries were made. One item represents a security vulnerability with *Low* impact score, and the other is a general weakness, also marked by little-to-no exploitation potential. This is a very good result for the Passbolt team, which has provenly managed to navigate around the usual issues spotted commonly in password management browser extensions.

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs. Subsequently, a chapter that sheds light on the test coverage reached by Cure53 is included to show what was looked at despite no findings spotted. Next, both findings will be discussed. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this April 2021 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team’s observations and collected evidence. Tailored hardening recommendations for the Passbolt complex are also incorporated into the final section.

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

Scope

- **White-Box Penetration-Tests & Audits against Passbolt Browser Extension(s)**
  - **WP1**: Penetration-Tests & Code Audits against Passbolt Chrome Extension
    - https://github.com/passbolt/passbolt_browser_extension/tree/master/dist/chrome
  - **WP2**: Penetration-Tests & Code Audits against Passbolt Firefox Extension
    - https://github.com/passbolt/passbolt_browser_extension/tree/master/dist/firefox
  - **Sources**
    - All relevant sources are available as OSS
    - https://github.com/passbolt/passbolt_browser_extension

Test Methodology

This section briefly summarizes Cure53's testing process in order to transparently describe the overall coverage achieved in this pentest against the Passbolt Browser Extensions. The following notes highlight steps taken to make sure it is understood which sensitive areas were explored, as well as which security bug classes were covered during this audit. The section begins with an analysis of the frontend JavaScript code and later moves to security tests against the utilized cryptography and other parts of the scope.

Audit of frontend JS code and Browser Extension vulnerabilities

- Cure53 examined the properties of the *manifest.json* file for Chrome and Firefox. Files exposed in *web_accessible_resources* were checked to ensure they cannot be leveraged as a Clickjacking vector. Although *config-debug.html* intended for development use was found to be "embeddable", generally all exposed HTML files are handled correctly. They cannot be directly embedded as they require to be spawned from the background script.
- Next, *externally_connectable* was confirmed to be disabled, which is correct because it could allow websites to communicate with the extension. Similarly, *content* scripts are not configured; instead, they are dynamically injected via *tabs.executeScript* which provides more granular controls. *Permissions* were also checked to ensure unnecessary items are not requested.
- Moving on, XSS possibilities in the extension were examined. Due to the use of the React framework, most traditional XSS vectors have been eliminated. The remaining vector, which is the misuse of *dangerouslySetInnerHTML*, was not found in the codebase and deemed secure. It is worth noting that CSP was not relaxed, meaning that even in the event of an XSS, exploitability would be very unlikely.
- In addition, XSS via the extension to websites was attempted. The code for injecting username and password onto a page only ever interacts with the DOM provided by the

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

isolated *content* script context and did not create any additional elements. Hence, this is considered a safe approach.

- Cure53 put emphasis on auditing the URL suggestion and insertion logic which are critical for password managers. The fact that autofill and saving password after logging in a web page were not supported significantly reduces the attack surface. DOM Clobbering and iframe abuse were tested to see if they can confuse the extension, but they are properly handled as well.
- All URLs to be validated go through normalization via the *window.URL* API which almost guarantees no discrepancies or parser differentials. Different URL components are correctly compared. IPv6 and different representation of IP (e.g. decimal) was also taken care of, with the exception of hostnames without a dot which may potentially cause confusion (see PBL-02-001).
- Finally, the communications between *background* script, *content* script and webpage have been examined. Message interception or spoofing were not possible. API interactions with Passbolt Cloud were also fine.

Audit of utilized Cryptography and related Parts

- Cure53 examined the properties attributed to the extension's environment as it is loaded into the Chrome and Firefox browsers. The importance of the correct contribution of these attributes is accentuated by the fact that the application will be storing sensitive cryptographic keys both locally and in memory.
- Passbolt implements a common OpenPGP interface through a collection of thoroughly documented and well-specified JavaScript classes, linked to a set of models that then construct a top-level React application. This common OpenPGP interface was checked for sanity, with special focus on PGP payload parsing operations, key generation, key management, key storage, passphrase management and storage, as well as batch encryption and decryption functionality.
- Passbolt provides certain network functionalities that involve a protocol for information exchange and authentication with a server. OpenPGP is used for the generation of *authorization* tokens. This logic was checked for sanity and absence of potential replay attacks or state machine-level attacks.
- The Passbolt extension offloads virtually all sensitive cryptographic operations onto the low-level PGP layer. This leaves the extension with a relatively small attack surface. It is rendered even smaller by the application's strict adherence to the React application discipline, with well-specified JavaScript modules all connected to the OpenPGP API.
- Finally, testing was conducted in order to verify the potential for user error resulting in degraded security.

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *PBL-02-001*) for the purpose of facilitating any future follow-up correspondence.

PBL-02-001 Extension: *canSuggestUrl* might suggest TLD entries *(Low)*

Passbolt suggests credentials based on the current tab's URL. It does so by comparing the URL's protocol, hostname and port against a list of stored URLs. In addition, it also compares if the current hostname is a subdomain of a stored URL.

Affected File:
all/data/js/quickaccess/popup/components/HomePage/canSuggestUrl.js

Affected Code:

```
// Otherwise check if the suggested url hostname is a parent host of the url
hostname.
  return isParentHostname(suggestedUrlObject.hostname, urlObject.hostname);
```

In the *isParentHostname* function, it returns *true* if *parent* appears at the end of *child*, with a caveat that it has to follow a dot or nothing. While the validation is sound against typical URL confusion attacks, there are two scenarios that this check misses.

The first one is when a user has a stored entry on an Intranet address. For example, if the user stores credentials for https://email pointing to the company's email system, then Passbolt will suggest that entry when the user visits something like https://titan.email/ because it has the same gTLD.

The second scenario is that some TLDs are directly accessible from the Internet at the *root* level. This can be seen in http://ai/ (works on MacOS). If a user stores credentials on these websites, the same thing will happen.

Although these are edge cases, it is recommended to consider not to suggest entries when the current URL does not contain a dot. A full origin matching should be performed in this case.

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

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals 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.

PBL-02-002 Web: Lack of CSRF protection on *logout* (*Info*)

It was found that the *logout* endpoint for Passbolt Cloud does not have a CSRF protection. If a *login* endpoint was also vulnerable to CSRF, then an attacker would be able to chain the flaws together. In effect, whenever a victim created a new password entry, it would be stored on the attacker's account instead. For now it can only cause annoyance to users.

Steps to reproduce:

1. Be logged into Passbolt Cloud
2. Navigate to https://cloud.passbolt.com/$namespace/auth/logout.json
3. You will be logged out

Alternatively, the *logout* URL can be embedded in a <*img*> tag for stealthiness. It is recommended to implement a CSRF protection on the affected endpoint as a defense-in-depth mechanism.

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

Conclusions

As noted in the *Introduction,* the Passbolt team has mastered the topics of security and privacy in both their Chrome and Firefox browser extensions. After dedicating eight days to testing in April 2021, three members of the Cure53 team only spotted flaws that would have low-relevance and/or seem very hard to exploit. It must be emphasized that this is by no means a typical result. In fact, the attack surface of projects in the Passbolt's realm of operations is usually quite large, given what and how the extensions in scope promise and deliver in terms of security. In the case of the Passbolt Browser Extensions, the risks associated with malicious websites confusing the extension and exploiting them have been largely mitigated.

To first comment on cryptography, the Passbolt extension's crypto-architecture comprises a comprehensive wrapper around OpenPGP, built as a set of classes that constitute models for a top-level React application. The OpenPGP wrapper was found to be highly comprehensive, well-specified, properly deployed with a uniform and consistent implementation strategy. All those properties further reduced the attack surface, which was already small due to the chosen cryptographic architecture.

The extensions are loaded into the Chrome and Firefox browsers with correct sandboxing and permission-handling. The Passbolt's network calls for client-server operations were equally well-specified and relied on OpenPGP for low-level cryptographic operations, once more reducing possibilities for vulnerabilities. Despite a thorough review, no cryptographic vulnerabilities could be spotted in the Passbolt Browser Extensions.

Moving on to extensions themselves and the related JavaScript, it should be clarified that the WebExtension audit started with checking the *manifest.json* for both Chrome and Firefox versions. Files exposed via *web_accessible_resources* were checked to see if they can be abused (e.g. via Clickjacking). The handling was found secure as all HTML files require injections through the *background* script.

Other properties, namedly *externally_connectable, content_scripts, permissions* were also checked. Cure53 documented that these either have not enabled or can be considered safe. XSS in and via the extension was attempted but the use of the React framework for the UI eliminates a vast collection of XSS vectors. The misuse of *dangerouslySetInnerHTML* was not found in the code and, hence, no XSS in the extension was spotted. Similarly, the *content* script does not inject additional DOM elements other than simulating cursor and keyboard events. In that sense, no XSS via the extension was possible.

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

The main focus was put towards examining the URL suggestion and inserting logic. Thanks to not supporting auto-fill and password saving in web pages, many attacks were fended off by default. Regarding URL suggestion, URL validation was checked and it correctly handles various URL quirks involving IP addresses. However, under certain circumstances, it could be tricked to suggest credentials stored for other websites (see PBL-02-001).

Other possible attack vectors like DOM Clobbering and iframes of different origins were checked but no issues were found. Finally, the communication between the *background* script, *content* script and the webpage was confirmed as safe. Messages are immutable, remaining safe when it comes to interception or spoofing. API interactions with Passbolt Cloud are also fine.

All in all, Passbolt WebExtension gives a good impression in terms of both code quality and security. Similarly positive verdict can be maintained for the implementation of the already audited cryptography. The Passbolt extension stands strong and the audit and pentest did not manage to unveil any serious severity bugs, whereas the overall number of problems is also limited to just two minor flaws. This is a very good result, especially after a rather high number of findings exposed in *PBL-01*. It is apparent that the development team has a good grasp of both the web and browser security and cryptography. This comes as no surprise given the vast experience they have gained through past projects. From the perspective of security and privacy, Passbolt can be judged as a praiseworthy, production-ready browser extension.

Cure53 would like to thank Remy Bertot and Thomas Oberndörfer as well as the rest of the Passbolt team for their excellent project coordination, support and assistance, both before and during this assignment.