Fine penetration tests for fine websites

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

Pentest-Report Passbolt UWP Windows App 03.2024

Cure53, Dr.-Ing. M. Heiderich, M. Pedhapati

Index

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

Introduction

“Finally, a password manager built for organizations that take their security and privacy seriously. Passbolt is trusted by 15 000 of them worldwide, including F500 companies, the defense industry, universities, startups and many others.”

From https://www.passbolt.com/

This report describes the results of a short best-effort penetration test and source code audit against the Passbolt UWP Windows application. The work was requested by Passbolt SA in February 2024 and performed by Cure53 in March 2024, in CW12. A total of two days were invested to achieve the expected coverage for this project.

The work was structured according to a single work package (WP), which is as follows

- **WP1**: Source code audits against the Passbolt UWP Windows application

Cure53 was provided with the source code, test-supporting documentation, and any other access required to perform the tests, using a white-box methodology. A team of two senior testers was assigned to prepare, execute and close this project. All preparations were made in March 2024, CW11, so that Cure53 could have a smooth start.

Communication during the test was done via a dedicated shared Slack channel between the Passbolt and Cure53 teams, to which all involved personnel from both parties were invited. Communication was smooth and there were not many questions to be asked, the scope was well prepared and clear, and there were no significant roadblocks during the test. Cure53 provided frequent status updates on the test and related findings, and live reporting was provided by Cure53 via the aforementioned Slack channel.

Due to the best-effort nature of this audit and the resulting time constraints, the Cure53 team was able to achieve decent coverage of the scope items. Nevertheless, the team was able to identify a total of five findings, four of which were classified as security vulnerabilities and one as a general weakness with a lower potential for exploitation.

This security audit of the Passbolt UWP Windows application revealed a strong security foundation implemented by the development team. The codebase is of high quality overall, and the chosen architecture and frameworks promote inherent resilience. Nevertheless, Cure53 was able to identify several vulnerabilities that need to be addressed quickly. While it is positive that no high or critical severity findings were identified, it is still recommended that these issues be addressed as soon as possible, as this will significantly improve the security posture of the application and strengthen its overall robustness and reliability.

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

Finally, it should be noted that this audit was conducted as a best effort test, as the time frame was rather short. However, some aspects of the scope have not yet been reviewed in detail, particularly the application's JavaScript code, and it is therefore recommended that these aspects be re-audited in the near future to ensure that any potential vulnerabilities or weaknesses are uncovered. The report will now go into more detail about the scope and setup of the test, as well as the material available for testing.

After that, the report will list all findings in chronological order, first the discovered vulnerabilities and then the common vulnerabilities discovered in this test. Each finding will be accompanied by a technical description, a PoC where possible, and mitigation or fix advice. The report will then conclude with a summary in which Cure53 will elaborate on the general impressions gained throughout this test and share some words about the perceived security posture of the target, which is the Passbolt UWP Windows application.

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

Scope

- **Source code audits & penetration tests against the Passbolt UWP Windows app**
  - **WP1**: Source code audits against the Passbolt UWP Windows application
    - **Sources:**
      - https://github.com/passbolt/passbolt-windows
    - **Commit:**
      - *D36ea22226c49c45e35abd2254eea11def3c67d0*
    - **Documentation:**
      - Detailed test-supporting documentation was shared with Cure53
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

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

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, all tickets are given a unique identifier (e.g., PBL-11-001) to facilitate any future follow-up correspondence.

PBL-11-001 WP1: Insecure Regex pattern allows canNavigate bypass *(Medium)*

During source code review, a bug was found in the construction of the *allowedUrls* list that uses *currentUrl*. The problem occurs because the dots in *currentUrl* are not properly escaped. This allows any character to be substituted in place of the dot, potentially resulting in incorrect or insecure URL pattern matching.

For example, if *this.currentUrl* is set to *UUID.ai.passbolt.local*, it's possible to bypass the *canNavigate* restriction with a URL like *UUID.ai?passbolt.local*. This happens because the regex designed to match *allowedURLs* doesn't take into account the unescaped dots, allowing variations of the URL that should not be allowed.

The snippets below show the affected code, where *currentUrl* is used to construct the regex in both background and rendered webviews.

Affected file #1:
passbolt-windows/passbolt/Services/NavigationService/BackgroundNavigationService.cs

Affected code #1:
```
 public void Initialize(string currentUrl)
        {
            this.currentUrl = currentUrl;
            string pattern =
$"^https://{this.currentUrl}/Background/(index-import\\.html|index-
auth\\.html|index-workspace\\.html)$";

            base.allowedUrls = new List<Regex>()
                {
                    new Regex(@pattern),
                };
        }
```

Affected file #2:
passbolt-windows/passbolt/Services/NavigationService/RenderedNavigationService.cs

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

Affected code #2:

```
 public void Initialize(string currentUrl)
        {
            this.currentUrl = currentUrl;

            string pattern = $"^https://{this.currentUrl}/Rendered/(index-
import\\.html|index-auth\\.html|index-workspace\\.html)$";

            base.allowedUrls = new List<Regex>()
        {
```

The problem can be reproduced by running the following Javascript code in the renderer's devtools.

PoC:

```
location.href='https://fcfea485-fbde-4acb-94c4-27a2b101573a.ai?
passbolt.local/app/passwords'
```

To work around this vulnerability, it is recommended to escape the dot in the *changeUrl* function. This adjustment ensures that the dot is treated as a literal character in the regex, rather than as a wildcard, thereby increasing the accuracy and security of the URL matching process.

PBL-11-002 WP1: PasswordVault can be accessed by Desktop apps *(Low)*

It was discovered that the PasswordVault, which is used to store user configurations, metadata, and secrets, is accessible by other desktop applications. This finding challenges the claims in the provided threat model that it is not possible to extract information from PasswordVault.

However, a review of the official documentation revealed that regular desktop applications, excluding *Appcontainer* applications, can indeed access the information in PasswordVault, contradicting the initial threat assessment.

Affected file:
passbolt-windows/passbolt/Services/CredentialLockerService/CredentialLockerService.cs

Affected code:

```
using Windows.Security.Credentials;

namespace passbolt.Services.CredentialLocker
{
    public class CredentialLockerService
    {
        private PasswordVault vault;
        private LocalUserManager localUserManager;
```

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

```
        public CredentialLockerService()
        {
        this.vault = new PasswordVault();
[...]
```

Because of this vulnerability, it is recommended to use the Data Protection API (DPAPI) to store user-sensitive information. DPAPI provides a more secure method of data protection because it provides user- and system-specific encryption capabilities that increase the security of stored data against unauthorized access by other applications.

PBL-11-003 WP1: JS execution by modifying LocalFolder Resources *(Low)*

While investigating access control issues, a discovery was made regarding the interaction between the background and renderer processes in the application. The use of webview's *SetVirtualHostNameToFolderMapping*[1] to run the UI from a local folder index was examined. It was discovered that the JavaScript resources in this local folder are configured with permissions that allow the currently logged on user to modify these files. Such modifications could lead to JavaScript execution within the Passbolt Background and Rendered UIs.

Although exploiting this issue requires the adversary to have physical access to the local computer, it is still a significant security concern. To mitigate this vulnerability, it is recommended that the permissions of these JavaScript files be changed to read-only. This change would prevent unauthorized modification of the files, thereby protecting the application from potential malicious code execution from these resources.

PBL-11-005 WP1: Arbitrary requestId used as topic in background IPC *(Medium)*

During the audit of the Inter-Process Communication (IPC) flow facilitated by *WebMessageReceived*, a vulnerability was identified in the messaging between *webviewRendered* and *webviewBackground*. Specifically, IPC messages sent from *webviewRendered* to *webviewBackground* can inappropriately send arbitrary WebMessage topics from the background, resulting in the invocation of privileged WebMessage handlers intended only for *webviewBackground*.

The root of the problem lies in the handling of *requestId*. The *requestId* sent by *webviewRendered* to *webviewBackground* is repurposed as the topic value for responses from *webviewBackground*.

[1] https://learn.microsoft.com/en-us/dotnet/api/microsoft.web.[...]-dotnet-1.0.2365.46

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

Since it is possible to set an arbitrary *requestId*, an adversary with access to *webviewRendered* via XSS could, for example, send a message with "requestId" like this:

PoC:
```
window.chrome.webview.postMessage(`{
    "topic": "passbolt.password-expiry.get-or-find",
    "requestId": "passbolt.auth.logout",
    "message": ""
}`)
```

This results in *webviewBackground* sending a response with the following message:

```
{topic: 'passbolt.auth.logout', status: 'SUCCESS', message: null}
```

Although the above postMessage from *webviewBackground* is a response to *webviewRendered* for the requestId *passbolt.auth.logout*, the main controller will treat it as a legitimate topic, which will then be executed incorrectly. The above postMessage PoC from *webviewRendered* will log out the user. However, the team couldn't find any cases with the message value control, hence the reduced impact.

Affected file:
passbolt-windows/passbolt/Webviews/Background/dist/background-auth.js

Affected code:
```
[...]
this.worker.port.emit(this.requestId, "SUCCESS", settings);
[...]
```

To address this vulnerability, it is recommended that the functionality of *requestId* and topic be clearly separated by using different attributes for each. This change ensures that the requestId sent by *webviewRendered* cannot be misused as a topic to perform privileged actions in *webviewBackground*By clearly delineating these attributes, the application can protect itself from unauthorized IPC message handling and improve overall security.

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

PBL-11-004 WP1: Insecure CSP Configuration in renderers *(Low)*

During the evaluation, it was discovered that the Content Security Policy (CSP) is not properly configured in both renderers. The problem can be seen in the provided code snippet where the critical content attribute of the meta tag is missing and the CSP value is incorrectly assigned without an attribute. In this context, *default-src* is incorrectly treated as an attribute. This syntax error causes the CSP directives not to be applied as intended.

Affected file:
passbolt-windows/passbolt/Services/LocalFolder/LocalFolderService.cs

Affected code:

```
 public async Task CreateRenderedIndex(string name, string script, string
stylesheet, string csp = null)
        {
            StorageFile indexFile = await this.CreateFile("Rendered",
name);
            var content = "<!DOCTYPE html> <html> <head>  <meta
charset=\"UTF-8\"> " +
              $@"<meta http-equiv=""Content-Security-Policy"" default-src
'self'; script-src 'self'; img-src 'self' {csp}; />" +
             [...]

         public async Task CreateBackgroundIndex(string name, string
script, string csp = null)
        {
            StorageFile indexFile = await this.CreateFile("Background",
name);
            var content = "<!DOCTYPE html><html> <head>  <meta
charset=\"UTF-8\"> " +
              (csp != null ? $"<meta http-equiv=\"Content-Security-Policy\"
default-src 'self' ${csp} https://api.pwnedpasswords.com; script-src 'self'
/></head>" : "") +
        [...]
```

To address the vulnerability, it is recommended to correctly assign the CSP value to the content attribute of the meta tag. This adjustment will ensure the proper enforcement of the CSP directives, enhancing the security of the application.

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

Conclusions

Overall, the Passbolt UWP Windows application made a robust impression in terms of its security posture. This is also reflected in the number of issues outlined in this report, as only low and medium severity vulnerabilities were found.

A white-box testing methodology was used to assess the specified areas of the application. This assessment confirmed that the Passbolt UWP Windows application development team successfully avoided and mitigated a number of typical desktop applications built on top of *Webview2*.

An established Slack channel facilitated productive communication and seamless information sharing between the teams. The clearly defined scope of the assessment and a shared understanding of the target areas minimized potential roadblocks throughout the process.

The Cure53 team was provided with a Passbolt Pro license for the application's API, necessary builds, documentation, and threat model to facilitate testing of the Windows application. The testing team effectively covered a significant portion of the scoped areas during their audit. However, it is important to note that due to time constraints, the team was unable to review the application's JavaScript code. With this in mind, it is recommended that future testing efforts be expanded to include a thorough review of both the UWP components and the JavaScript code to ensure a more comprehensive security assessment.

The testing process began with an examination of the use of *Json.NET* to identify potential deserialization issues. After a careful review, the team concluded that deserialization was performed securely using *SerializationBinder*, effectively mitigating the associated risks. The team then audited the application's handling of navigating to arbitrary URLs and opening new windows. This part of the audit resulted in the identification of a problem documented in PBL-11-001.

The audit included a thorough examination of the *AddWebResourceRequestedFilter* implementation, which restricts requests to only those hosts on the background renderer's allow list. The team did not find any problems or potential bypasses in this implementation. In addition, the use of PasswordVault to store sensitive information was reviewed. A minor issue was found in this area, which is detailed in PBL-11-002.

Finally, the IPC (Inter-Process Communication) interactions between the main controller, renderer, and background components were reviewed for possible misconfigurations. This review resulted in the identification of a specific problem that was documented in PBL-11-005.

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

The team conducted a detailed investigation of the potential attack vectors outlined in the documentation. During this exploration of various attack scenarios, several minor issues were identified and documented as PBL-11-003 and PBL-11-004.

In conclusion, upon completion of this security audit, Cure53 gained a strong impression of the security premise employed by the Passbolt team. The quality of the codebase was generally impressive, while the architecture and frameworks employed generally installed resilient design paradigms. Nevertheless, it is important to acknowledge the discovery of the issues detailed in the respective tickets. Addressing and resolving these reported issues will further strengthen the security posture of the application, increasing its robustness and reliability.

Cure53 would like to thank Pierre Colart, Stephane Loege, Maxence Zanardo and Remy Bertot from the Passbolt SA team for their excellent project coordination, support and assistance, both before and during this assignment.