Fine penetration tests for fine websites

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

Pentest-Report Proton Pass Browser Addon, Apps & API 05.-06.2023

Cure53, Dr.-Ing. M. Heiderich, M. Pedhapati, Dipl.-Ing. A. Inführ, L. Herrera

Index

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

Introduction

“Proton was born out of a desire to build an internet that puts people before profits, create a world where everyone is in control of their digital lives, and make digital freedom a reality. In this new world, you can communicate with whomever you want, protect your data and identity, avoid having your data sold, and safeguard against cybercrime.”

From https://proton.me/about

This report describes the results of a security assessment of the Proton complex, more specifically covering the Proton Pass mobile applications and browser addon, as well as the web application and backend API endpoints. The project, which included a penetration test and a wider review of the security posture, was carried out by Cure53 in May 2023.

Registered as *PRO-01*, the examination was requested by Proton AG in April 2023 and then scheduled to start the following month. Sufficient time was available for both preparations, which was particularly important in the context of this assignment marking the first cooperation between Cure53 and Proton.

In terms of the exact timeline and specific resources allocated to *PRO-01*, Cure53 completed the research in CW22 of 2023, namely in late May and early June. In order to achieve the expected coverage for this task, a total of twelve days were invested. In addition, it should be noted that a team of four senior testers was formed and assigned to prepare, execute, and deliver this project.

For optimal structuring and tracking of tasks, the examination was split into four separate work packages (WPs):

- **WP1**: White-box penetration testing against Proton Pass Android mobile application
- **WP2**: White-box penetration testing against Proton Pass iOS mobile application
- **WP3**: White-box penetration testing against Proton Pass browser addon
- **WP4**: White-box penetration testing against Proton Pass web & backend API

As the titles of the WPs suggest, white-box methodology was ultimately utilized during this *PRO-01* project. It should be noted that the originally requested methodological approach for this project entailed a gray-box premise. However, during the setup, it was decided that sources of the applications would be shared with Cure53. As such, the project’s methods were transformed into a white-box investigation and relied on the corresponding toolset and strategies.

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

Cure53 was provided with mobile application builds, test-supporting documentation, test-user accounts, as well as all other means of access required to complete the tests.

Overall, the project progressed effectively. To facilitate a smooth transition into the testing phase, all preparations were completed in CW21 Throughout the engagement, communications were conducted via a private, dedicated and shared Slack channel. Stakeholders - including the Cure53 testers and the internal staff at Proton AG - could participate in discussions in this space.

The quality of the interactions throughout the test was excellent, with no outstanding queries. Steady exchanges between the participating teams contributed positively to the overall outcomes of this project. The scope was well-prepared and clear, which played a major role in avoiding significant roadblocks during the test.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting of the spotted issues was completed via the shared Slack channel, which means that the Proton team could not only directly ask questions about the discoveries, but also had the option to start developing fixes while the Cure53 tests continued.

The Cure53 team succeeded in achieving very good coverage of the WP1-WP4 scope items. Of the ten security-related discoveries, eight were classified as security vulnerabilities and two were categorized as general weaknesses with lower exploitation potential. The overall number of findings should be seen as moderate and pointing to a proper state of security across the investigated Proton targets. Moreover, the majority of findings were ascribed with limited severities, which showcases the rather low level of exploitability attached to the unearthed risks.

Given that this project was the first time that the Proton Pass applications and their components were tested by Cure53, it is a positive surprise that not many issues transpired. There are two caveats, however, to this verdict. First, the project had a limited time budget in relation to the rather large expanse of the scope. This led to an unavoidable prioritization of the most sensitive features and functionalities.

Second, one of the findings needs to be explicitly mentioned, as it was ranked with a *High* severity and demonstrated an overall lack of public suffix checks (see PRO-01-006). In order to eradicate potential leakage of user-credentials, it is strongly recommended to swiftly mitigate this vulnerability.

The following sections first describe the scope and key test parameters, as well as how the WPs were structured and organized. Next, all findings are discussed in grouped vulnerability and miscellaneous categories.

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

Flaws assigned to each group are then discussed chronologically. In addition to technical descriptions, PoC and mitigation advice will be provided where applicable.

The report closes with drawing broader conclusions relevant to this May-June 2023 project. Based on the test team's observations and collected evidence, Cure53 elaborates on the general impressions and reiterates the verdict. The final section also includes tailored hardening recommendations for the Proton Pass mobile applications and browser addon, as well as the web application and backend API endpoints.

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

Scope

- **Penetration tests & Security assessments against Proton Pass apps, addon & API**
  - **WP1**: White-box penetration testing against Proton Pass Android mobile app
    - **Build:**
      - *pass-android.b2e85ed8d.apk*
  - **WP2**: White-box penetration testing against Proton Pass iOS mobile app
    - **TestFlight:**
      - https://testflight.apple.com/join/yZUVAr0N
  - **WP3**: White-box penetration testing against Proton Pass browser addon
    - **Build:**
      - *pass-browser-extension.build.3925b4afed.tgz*
  - **WP4**: White-box penetration testing against Proton Pass web & backend API
    - **API documentation was shared with Cure53**
  - **Test-user-accounts:**
    - U: passaudit01@proton.me
    - U: passaudit02@proton.me
    - U: passaudit03@proton.me
  - **Test-supporting material was shared with Cure53**
  - **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

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., *PRO-01-001*) to facilitate any future follow-up correspondence.

PRO-01-002 WP3: Overly permissive *externally_connectable* property *(Low)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

While auditing the extension file for any misconfigurations, it was noticed that the *externally_connectable* property had many wildcard domains listed. This lets a malicious adversary read the items in the Proton Pass, realizing risks linked to XSS or subdomain takeover on any of these domains. However, the Cure53 team could not find any instances of XSS or subdomain takeovers in the *externally-connectable* domains, which explains the reduced impact-score.

Affected file:
manifest.json

Affected code:

```
[...]
    "externally_connectable": {
        "matches": [
            "https://*.proton.me/*",
            "https://*.proton.local/*",
            "https://*.proton.black/*",
            "https://*.proton.pink/*"
        ]
    },
[...]
```

The following PoC shows a way to leak user-credentials from the specified domains.

PoC:

```
await chrome.runtime.sendMessage('ghmbeldphafepmbegfdlkpapadhbakde',
{"type":"AUTOFILL_SELECT","payload":{"shareId":"IP1djJdd0za_ZJJIpf-
I58moogKkhQazO3ptbLvEBqmrGiVpLh2N5fAwthtrZHfC4ibImNSUafI71CghVh8IXg==","itemId":
"0FvEW2R_6rpvZWwoKjNJ7a7TlxF8Skjsg0tC7Gf2xyEFJGmWr_ZkQTsd8zP2t8iOz126xgru8Gd_48p
PwooMNQ=="},"sender":"content-script"},console.log)
```

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

It is recommended to ensure that the *externally_connectable* property only contains the domain that is needed, in this case *account.proton.me*. This will help to reduce the attack surface on the extension.

PRO-01-003 WP1: Autofill can be invoked from within iframes *(Medium)*

***Note**: Proton has discovered that this issue cannot be resolved at this time due to a platform limitation in Android (the Android operating system does not currently provide the information that would be required to resolve this issue).*

The observation was made that the *autofill* feature from the Android application could be used from within cross-origin iframes. After further investigation, it was discovered that the hostname of the top-level page is being used to determine the suggested credentials. This is deployed instead of the hostname of the iframe that initiated the *autofill* request. The top-level URL of the page is also displayed in the Android application when one chooses the *Open Proton Pass* option.

Both of these behaviors are problematic, given they can be abused to trick users into auto-filling their credentials in the context of malicious pages. This happens in the case that the targeted page contains third-party iframes commonly used for advertisements or analytics, for example.

Steps to reproduce:

1. Install the Proton Pass Android application and log in.
2. Add any credentials to the *https://proton-test.tiiny.site* website.
3. Access *https://proton-test.tiiny.site/* and click on the *password input* field.
4. Select the credential that will be displayed and
5. Observe an alert containing the email address and password.

To mitigate this issue, Cure53 advises preventing the user from auto-filling their credentials to iframes, similar to what has been done in the Proton Pass extension. Another option would be to use the iframe's *src* attribute value to suggest the available credentials to be auto-filled. Additional guidance on this matter can also be found in Android's *Autofill's Web security* documentation's section[1].

[1] https://developer.android.com/reference/android/service/autofill/AutofillService#web-security

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

PRO-01-004 WP3: Items viewable without PIN code despite lock *(Low)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The password and other items of a given Proton Pass extension are not cleared from memory immediately after the user locks the application. This means that an attacker with physical access to the victim's computer can retrieve the victim's saved items, even if the extension is locked. Notably, the locking mechanism only takes effect on the server-side, while no previous data is cleared from memory.

Steps to reproduce:

1. Ensure Google Chrome is used and the Proton Pass extension is installed.
2. Add PIN lock in the *settings* and lock the extension.
3. Go to *chrome://extensions/* and open *Devtools* of the service worker.
4. Click on the *Memory* tab on the appearing *DevTools* window.
5. Click on *Take snapshot*.
6. Press *CTRL+F* and type any username or password stored in the Proton Pass extension after the snapshot has been created.
7. Observe that the password is identifiable in the memory.

To mitigate this vulnerability, one should ensure that the Proton Pass extension is closed after the lock, so that the memory is sufficiently cleared.

PRO-01-005 WP1: URL verification fails to compare protocols *(Low)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The domain verification of *login* items and the currently loaded URL was checked. This unveiled that the Android application is not comparing the specified protocols. In case a *login* item is stored for *ftp://example.com*, it will match *http://example.com*.

This behavior is not present for the iOS application, as it is matching the specified protocols. Still, the impact of this behavior is rated *Low,* as in modern browsers the same *origin policy* prevents the protocol mismatch from being exploitable.

Affected file:
ProtonPass/pass/data/impl/src/main/kotlin/proton/android/pass/data/impl/autofill/SuggestionItemFilterer.kt

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

Affected code:

```
private fun isDomainMatch(urlDomain: String, itemDomains: List<String>): Boolean
{
       [...]
              is HostInfo.Host -> when (parsedDomain) {
                     is HostInfo.Host -> {
                            parsedDomain.tld == it.tld && parsedDomain.domain ==
                                   it.domain
                     }
                     else -> false
              }
```

It is recommended to compare the specified protocols as well, as it is already done in the implementation of the iOS application. This ensures that no potential information leaks can occur via abuse of special protocol handlers in web browsers.

PRO-01-006 WP3: Lack of public suffix check leads to credential leakage *(High)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The Proton Pass extension is not taking into consideration the public suffix list when deciding which credentials to suggest to the user in the context of auto-filling forms.

The current implementation is displaying the credentials associated with the *victim.github.io* hostname as an *autofill* suggestion for the *attacker.github.io* hostname. Though the *github.io* domain is in the public suffix list, it should not be permitted.

This would allow an attacker-controlled page hosted in a subdomain of a domain included in the public suffix list to leak the credentials of the users. Additionally, the severity of this issue is increased via clickjacking attacks, which happens by overlaying the extension's *dropdown.html* iframe and tricking the user into making unwanted actions.

Steps to reproduce:

1. Install the Proton Pass extension and log in.
2. Add any credentials to the *https://admin.github.io* website.
3. Access *https://lbherrera.github.io/lab/proton/index-31877327.html* and click on the Proton Pass icon next to the *username input* field.
4. Select the credentials
5. Note that an alert displaying the email address and password appears.

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

To mitigate this issue, Cure53 advises checking whether the hostname of the page is included in the public suffix list[2]. If so, Cure53 credentials should only be suggested for an auto-fill if they exactly match that particular hostname.

PRO-01-007 WP1: *HostParserImpl* in Android does not check for FQDN *(Medium)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The *HostParserImpl*'s parse function is not taking into consideration that a website can also be accessed using its fully qualified domain name (FQDN) format. This is done by adding a trailing *dot* to the end of the hostname.

The current parser's behavior mistakenly shifts the parts of a hostname, causing the TLD to be blank. The real TLD can be set as the domain, while the domain can be considered in the subdomain when a hostname in the FQDN format is utilized.

To exemplify, when the *https://github.io./* URL is parsed by the *HostParserImpl*'s parse function, the *HostInfo* data is set, as shown below.

***HostInfo* data:**

```
HostInfo.Host(
     subdomain = "github",
     domain = "io",
     tld = ""
 )
```

This is problematic because any URL added to the Proton Pass application with FQDN will be considered to contain the same domain, namely the TLD. This allows any domain that has the same TLD to auto-fill forms via the user's credentials.

Affected file:
/pass-android/ProtonPass/pass/data/impl/src/main/kotlin/proton/android/pass/data/impl/url/HostParserImpl.kt

Affected code:

```
private fun hostWithoutTld(parts: List<String>): HostInfo.Host {
    // We did not find a TLD, so we'll just assume that:
    // - The last portion is the tld
    // - The second to last is the domain
    // - The rest is the subdomain
```

[2] https://publicsuffix.org/

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

```
    val tld = parts.last()

    val hostInfo = if (parts.size > 2) {
        val subdomain = buildString {
            for ((portions, i) in (0 until parts.size - 2).withIndex()) {
                if (portions > 0) append('.')
                append(parts[i])
            }
        }


        HostInfo.Host(
            subdomain = subdomain.some(),
            domain = parts[parts.size - 2],
            tld = tld.some()
        )
    }
    [...]
}
```

Steps to reproduce:

1. Install the Proton Pass Android application and log in.
2. Add any credentials to the *https://victim.io./* website.
3. Access *https://lbherrera.github.io./lab/proton/index-31877327.html* and click on the *password input* field.
4. Select the credentials that will be displayed and the *input* field will be auto-filled.

To mitigate this issue, Cure53 advises checking whether the hostname ends with a trailing *dot* and removing it before parsing the URL in *HostParserImpl.*

PRO-01-008 WP3: Arbitrary port connections in *IFrameContextProvider* *(Medium)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

While auditing the Proton Pass extension mechanism for message passing, it was noticed that the message shifts from the *autofill dropdown* page and the content script of the current page takes place. This happens via the service worker using a random *port name* created by the content script.

This *port name* is passed to the *dropdown* page via *postMessage* from the content script. It was discovered that this *port name* can be changed to a victim's page port by running a crafted JavaScript in the attacker's page. This item races the actual *port name* from the content script by sending continuous *postMessages* to the *dropdown* iframe.

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

The issue effectively allows listening on the sensitive messages sent to the victim's page. However, as the *port name* is randomly generated, the Cure53 team could not find a way to fully exploit this issue to read the victim's *autofill* data. As such, the impact has been downgraded.

The following snippet shows the affected code where the *port name* is retrieved via *message listener* without any origin check.

Affected file:
applications/pass-extension/src/content/injections/apps/context/IFrameContextProvider.tsx

Affected code:
```
const portInitHandler = safeCall((event:
MessageEvent<Maybe<IFrameMessageWithSender>>) => {
    if (
        event.data !== undefined &&
        event.data?.type === IFrameMessageType.IFRAME_INJECT_PORT &&
        event.data.sender === 'content-script'
    ) {
        window.removeEventListener('message', portInitHandler);

        /* Open a new dedicated port with the worker */
        const message = event.data;
        const framePortName = `${message.payload.port}-${endpoint}`;
       //framePortName is retrieved via message listener
        framePortRef = browser.runtime.connect({ name: framePortName });
```

The following snippet shows the affected code where the background JavaScript service worker passes the data from the content script to the attacker's *dropdown* page.

Affected file:
applications/pass-extension/pass/extension/message/message-broker.ts

Affected code:
```
port.onMessage.addListener(async (message: Maybe<PortFrameForwardingMessage>,
source) => {
            if (message && message.type ===
WorkerMessageType.PORT_FORWARDING_MESSAGE) {
                if (ports.has(message.forwardTo)) {

ports.get(message.forwardTo)!.postMessage({ ...message.payload, forwarded:
true });
                } else {
```

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

```
                    source.postMessage(backgroundMessage({ type:
WorkerMessageType.PORT_UNAUTHORIZED }));
                }
            }
        });
```

The PoC script below shows how to connect to an arbitrary port named *content-xx-dd*.

PoC:

```
<form> <label for="username">Username:</label> <input type="text" id="username"
name="username" autocomplete="username" required> <label
for="password">Password:</label> <input type="password" id="password"
name="password" autocomplete="current-password" required> <input type="submit"
value="Login"> </form>
<script>window.onmessage=console.log;setInterval(function(){try{
frames[0].postMessage({"type":"IFRAME_INJECT_PORT","payload":{"port":"content-
script-xx-dd"},"sender":"content-script"},"*")}catch{};},2)</script>
```

It is recommended to pass the *port name* from the service worker instead of the page's content script. This disallows the page from modifying the *port name* that the *dropdown* connects to. If this is not possible, it is suggested to make the *port name* even more random.

PRO-01-009 WP3: *Autofill* on insecure HTTP origins *(Medium)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

Testing confirmed that the Proton Pass Chrome extension performs auto-filling on the page served via insecure *http: URL*. Specifically, the check that occurs prior to the *autofill* confirms whether the host constitutes the same domain or a subdomain, but it does not confirm the protocol.

As a result, by navigating the victim to the domain's *http: URL*, for which the credentials are stored in the extension, there is an increased risk connected with an attacker capable of performing a Man-in-the-Middle attack. The attacker positioned in this way could obtain the victim's credentials entered via the *autofill* feature. The issue can be reproduced via the following steps.

Steps to reproduce:

1. Open a *login* page served on *https:*.
2. Save the credentials to the Proton Pass extension.
3. Open a *login* page served on *http:* in the same domain.

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

4. Click on the *input* field of the *login* form. The *dropdown* list to select the saved credentials will be displayed. If the saved credentials from *Step 2* are selected, *autofill* will function.

To mitigate this issue, Cure53 recommends displaying a warning before entering the credentials and performing the *autofill* operation in the context of *http: URLs*.

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 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.

PRO-01-001 WP2: iOS Data Protection entitlement not fully utilized *(Info)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The iOS mobile application does not currently set a value of the *com.apple.developer.default-data-protection* entitlement, which controls the minimum Data Protection level for its data. Therefore, the default value *NSFileProtectionCompleteUntilFirstUserAuthentication* is used, granting continuous access to app-files if the phone is unlocked once.

To take full advantage of this feature, the *NSFileProtectionCompleteUnlessOpen* or *NSFileProtectionComplete* entitlement should be set. In case the app requires to fetch data while the phone is locked, *NSFileProtectionCompleteUnlessOpen* should be used.

The latter grants access to files that have already been opened by the app when the phone was locked, therefore allowing it to store additional data. In case no such background functionality is necessary, *NSFileProtectionComplete* can be used, as it prevents reading or writing of all the app-files while the phone is locked[3].

PRO-01-010 WP1: Unrestricted dangerous schemes *(Info)*

***Fix note**: This issue was addressed by the development team and the fix was successfully verified by Cure53. The issue as described no longer exists.*

The scheme of the URL added to a vault is not being validated against an allow-list of dangerous schemes set in the Android application. This is problematic because the same checks are being properly applied by the Proton Pass extension, which currently relies on the assumption that the Android application also does so. The mismatch causes one instance of stored XSS in the extension via the acceptance of the JavaScript scheme in the URL. This is subsequently added to the *anchor* tags.

[3] https://developer.apple.com/documentation/bundleresources/entitlements/com_[...]-data-protection

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

Fortunately, the CSP applied to the extension mitigates this issue and prevents arbitrary JavaScript executions. Thus, the severity of this flaw has been appropriately downgraded.

Affected file:
/pass-android/ProtonPass/pass/data/api/src/main/kotlin/proton/android/pass/data/api/url/UrlSanitizer.kt

Affected code:

```
object UrlSanitizer {
    fun sanitize(url: String): Result<String> {
        if (url.isBlank()) return Result.failure(IllegalArgumentException("url
cannot be empty"))
        if (url.all { !it.isLetterOrDigit() })
            return Result.failure(IllegalArgumentException("url cannot be all
symbols"))

        // If it doesn't have a scheme, add https://
        val urlWithScheme = if (!url.contains("://")) {
            "https://$url"
        } else {
            url
        }

        return try {
            val parsed = URI(urlWithScheme)
            if (parsed.host == null) return
Result.failure(IllegalArgumentException("url cannot be parsed"))
            val meaningfulSection = "${parsed.scheme}://${parsed.host}$
              {parsed.path}"
            Result.success(meaningfulSection)
        } catch (e: URISyntaxException) {
            Result.failure(e)
        }
    }
```

Steps to reproduce:

1. Install the Proton Pass Android application and log in.
2. Create a *login* using *javascript://test* as the website.
3. Install the Proton Pass extension and log into the same account.
4. Select the credentials created in *Step 2*.
5. Click on the *javascript:* URI that will be displayed in the *Websites* section.
6. Check the *DevTools* to confirm JavaScript execution prevented by CSP.

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

To mitigate this issue, Cure53 recommends creating an allow-list containing all protocols deemed dangerous - such as *javascript:*, *file:*, and *data:* (among others) - and checking the scheme's URL against it.

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

Conclusions

Cure53 can conclude that the Proton Pass apps and components leave a rather positive impression in terms of security. Even though there are multiple areas, which require some more attention and work, it is hoped that fixing all ten issues spotted during this May-June 2023 project will elevate the already existing resilience against a multitude of severe attacks and threats.

Four members of the Cure53 team involved in this *PRO-01* project strongly advise continuous commissioning of external tests and retests. The maintainers have to make sure that the Proton Pass apps are holistically protected and benefit from in-depth hardening.

During this *PRO-01* examination the Proton Pass browser addon was assessed first. The testing team hoped to locate client-side-related security issues associated with XSS, *postMessage*, and prototype-pollution. It was noted that the inspected frontend for the most part utilizes the ReactJS framework, which offers a battle-tested escaping mechanism that prevents a plethora of XSS issues by default.

Furthermore, the presence of *dangerouslySetInnerHTML* was audited for erroneous use, since this can often incur XSS. Since the ReactJS framework does not supervise the URLs assigned to the *href* property of the HTML *anchor* tags, the source code was searched for these issues. Finally, any risk-laden instances of w*indow.open, window.location* and similar usage were inspected. Positively, no findings were identified in this area.

Next, the addon's *manifest.json* was audited for any misconfigurations such as exposed *web_accessible_resources*, insecure permissions, weak *content_security_policy* and misconfigured *externally_connectable*. This revealed PRO-01-002 and although the *content_security_policy* is not defined, the Chromium extensions by default utilize strict CSP.

The testing team then focused on the communication between the background script, content script, and the webpage via message channels and *postMessage.* This led to identification of the issue described in detail in PRO-01-008.

Particular attention was also paid to whether the *autofill* operates exactly as intended within the expected domain's scope. The fact that it may leak credentials to unintended domains was a concern in this realm. In relation to this, two issues were identified. Firstly, a moderate weakness that could facilitate credential leakage via Man-in-the-Middle attacks was identified due to the lack of the protocol check (see PRO-01-009).

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

Secondly, a major issue concerning the lack of a public suffix check, happening upon generating a list of available credentials associated with a given domain, was uncovered. The latter could lead to credential leakage, too (see PRO-01-006).

Moving on to the mobile applications of the Proton Pass, the general attack surface exposed by both mobile applications is kept to a minimum. On Android, no unnecessary activities, broadcast receivers or services are exposed. Additionally, no intent parsing is implemented by the main activity. Similarly, it was also found that no custom URL protocol handlers are shipped by either of the two apps.

The storage concept makes use of the available operating system keyrings to ensure proper encryption of user-data. The logs were checked for leaks of the *authentication* token, but no such instance was spotted during the assessment. That security was part of the development is also reflected by the mobile displays of website favicons. To ensure the security of the user, the API handling the retrieval of favicons does not support arbitrary domains, which prevents potential attacks via malicious favicon images. A minor improvement was recommended for the iOS app, as it did not take full advantage of the available file protection options (see PRO-01-001).

The main focus was set on the domain parsing and comparison of the apps' *autofill* features, as mistakes in this component could leak the users' login credentials to unintended origins. The implementation on iOS was found to be solid. It correctly makes use of the public suffix list to determine top-level domains. Additionally, it compares the specified protocols as well, which its Android counterpart fails to do (PRO-01-005). Another improvement for the Android application lies in the current domain parsing logic, which can be confused by appending a *dot* character to a domain (PRO-01-007).

During the investigation of the *autofill* feature in the Android implementation, it was found that the feature could be initiated from within iframes, but incorrectly used the top-level page's URL to determine which credentials to suggest. This led to PRO-01-003. The iOS counterpart was found to be appropriately using the iframe's URL, and was not affected by this problem.

It was noted that a few implicit security assumptions were made in regard to the parity of the implemented features across the iOS, Android, and extension versions. A missing scheme check in the Android version was discovered, which led to a mitigated XSS vulnerability in the extension. More information about this issue can be found in PRO-01-010.

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

Finally, in regard to the API, the endpoints explicitly associated with the Proton Pass application have taken center stage and, generally, held well to scrutiny. An XSS attack would be highly impactful when combined with PRO-01-002, so attempts were made towards uncovering XSS issues that might affect the API. Luckily, no vulnerabilities were found in this realm. Additionally, it was noted that the API is using a strict CSP, further mitigating various risks.

Brute-forcing attempts against sensitive endpoints (such as the login and session unlock features) were performed, but the application is properly restricting access after an excessive amount of requests. The endpoints were exhaustively tested for authorization and access control issues, but no problems of the sort were discovered.

All in all, the impression garnered by the Pass applications was positive. The documented issues can be easily addressed and will strengthen the deployed origin verification in regard to the *autofill* feature. In comparison, the browser addon left a rather mixed impression, as most of the issues, including that with *High* scores - could be located in this component. To ensure a safe and secure usage of the addon and all other components, Cure53 recommends swiftly addressing and resolving all issues as soon as possible.

Cure53 would like to thank Adrià Casajús, Son Nguyen, Justas Vilgalys, Thanh-Nhon Nguyen, Krzysztof Siejkowski, Carlos Quintana, Victor Hidalgo and Edvin Candon from the Proton AG team for their excellent project coordination, support and assistance, both before and during this assignment.