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

Pentest-Report 1Password Core 11.-12.2021

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, MSc. F. Fäßler

Index

Introduction

"Millions of customers and more than 100,000 businesses trust 1Password to keep their most important information safe. At 1Password we believe everyone deserves to be safe online. That's why we're building modern, accessible apps with privacy and security at their core."

From https://1password.com/company/

This report describes the results of a security assessment of the 1Password complex, particularly the 1Password core code and software. Carried out by Cure53 in late November and early December 2021, the project included a penetration test and a dedicated audit of the source code.

Registered as *1PW-18,* the project was requested by 1Password in early 2021 as part of the annual penetration testing plan. It was scheduled for the last quarter of 2021 to allow ample time for preparations on both sides. The project is the latest of a total of *eighteen* testing iterations that took place, explaining the test label *1PW-18*.

As for the precise timeline and specific resources, Cure53 completed the examination as scheduled, in CW47 and CW48. A total of fourteen days were invested to reach the coverage expected for this assignment, whereas a team of three senior testers has been composed and tasked with this project's preparation, execution and finalization. All members of the testing team were already familiar with the 1Password software compound via previous project work.

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

- **WP1**: 1Password core codebase & software, written predominantly in Rust
- **WP2**: High-level research questions asked by the 1Password Team

As mentioned, this test and audit is part of the yearly penetration testing and code auditing routine executed by Cure53 against several elements of the 1Password software compound. In this test iteration, it was planned to look at builds for several platforms, but, ultimately, the scope was changed because not all items planned for this testing round were fully ready for testing. In the end, the following areas were covered:

- Key focus on the new Windows Hello feature and
- Key focus on the Browser + CLI integration on Linux
- High-level questions regarding macOS client

Cure53 was, as usual for engagements between 1Password and Cure53 given access to a very detailed scope document explaining all areas of interest and access parameters. This was provided alongside sources, binaries, documentation and everything else that was needed. The above indicates that white-box methodology was the approach of choice for this project.

The project progressed effectively on the whole. All preparations were done in CW46 to foster a smooth transition into the testing phase. As usual, preparatory work on the 1Password team's side was exceptionally good. Over the course of the engagement, the communications were done using a private, dedicated and shared Slack channel set up between the respective workspaces of Cure53 and 1Password. New channel was set up and all involved personnel could join the discussions.

The discussions throughout the test were very good and productive and not many questions had to be asked. The scope was well-prepared and clear, greatly contributing to the fact that no noteworthy roadblocks were encountered during the test. It has to be stated that the 1Password team, as in other tests before, was very helpful and did whatever was necessary to make it possible for Cure53 to get good coverage over the scope with the chosen approaches. The assistance spanned fast answers to all questions, very quick turnaround times and generally excellent test-support. Ongoing interactions positively contributed to the overall outcomes of this project.

Cure53 offered frequent status updates about the test and the emerging findings. Live-reporting was done for several of the findings to allow the 1Password team an early review of the issues and more time to ask questions about the possible fixes to be developed and deployed.

The Cure53 team managed to get very good coverage over the WP1-WP2 scope items. Among five security-relevant discoveries, two were classified to be security vulnerabilities and three to be general weaknesses with lower exploitation potential. It can be deduced that the results of this project put 1Password in a very positive light. The number of issues is small and only one item was marked as *High.* Cure53 strongly believes that frequent testing contributes to 1Password maturing and being in a good state. Two of the findings assume malware to be in place to function, whereas others are rather limited in terms of actual risks. Note that one of those findings, see 1PW-18-003, is quite similar to an issue reported several months ago as *1PW-10-010*, so it could have potentially been avoided.

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. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in each group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this late 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 1Password complex are also incorporated into the final section.

Scope

- **Penetration tests and code audits against 1Password core code & software**
  - **WP1**: 1Password Core codebase & software, written predominantly in Rust
    - In scope were builds for several platforms and their general security properties.
    - Key features in scope for this assignment:
      - Newly implemented Windows Hello feature
      - Improvements of Browser Integration feature on Linux
      - Newly added integration with CLI
  - **WP2:** High-level research questions asked by the 1Password team
    - Q1: *"In what way do we not protect well against system processes running with the same privileges as the current user?"*
    - Q2: *"Can same-user privilege processes perform successful active attacks against the desktop apps?"*
    - Q3: *"Can same-user privilege processes learn anything about vault contents of the desktop application when it's locked?"*
    - Q4: *"In what way can we reasonably improve our resilience against attacks that use administrative privileges?"*
    - Q5: *"Does our application contain exploitable desktop/electron based vulnerabilities, potentially those that can be leveraged by shared vaults?"*
    - Q6: *"Is browser communication something that can be hijacked?"*
- **Test-user-accounts**
  - **Account 1:**
    - https://seba-1pw18.b5test.com/
    - U: seba@cure53.de
  - **Account 2:**
    - https://fabian1pw18.b5test.com/
    - U: fabian@cure53.de
- **All relevant binaries have been shared with Cure53**:
- **Detailed test-supporting material has been shared with Cure53**
- **All relevant sources have been shared with Cure53**

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. 1*PW-18-001*) for the purpose of facilitating any future follow-up correspondence.

1PW-18-003 WP2: Windows malware can trivially backdoor *.html* and *.js* (*High*)

Note*: Like many Windows applications, 1Password installs to a location in the user's local application directory, which comes with the important limitations noted. The developers are working on ways to make a protected installation which installs to the Program Files directory more widely available.*

For enterprise customers, 1Password already offer an MSI installer for enterprise deployments that installs to the Program Files directory and is not affected by this issue. For regular installs, 1Password is working on a solution that provides a protected install for as many users as possible. Unfortunately, this does come with certain trade-offs in the reliability and security of the 1Password for Windows' installation and automatic updates. 1Password wants to get those trade-offs just right before they roll out a fix.

During a previous engagement dedicated to reviewing the macOS client, it was noticed that malware can trivially backdoor the Electron files (see *1PW-10-010*). This was also confirmed to be the case on Windows. This happens when files are not installed into a trusted folder, as is currently the case. Malware can simply modify the HTML and JavaScript code of the 1Password client, making extractions of unlocked passwords trivial.

Steps to reproduce (on Windows):

1. Ensure that 1Password has been terminated
2. Go to the location of the *app.asar* file, i.e., *C:\Users\<Username>\AppData\Local\1Password\app\8\resources*
3. Unpack the application files with `npx asar extract app.asar app_unpacked`
4. The unpacked files are now written to the *app_unpacked* folder. To show the backdooring capabilities, simply create a *backdoor.js* file and load it in the *primary.html* via `<script src="backdoor.js"></script>`
5. The *backdoor.js* file will be executed when the 1Password app is opened. A simple `alert(1)` can prove the successful JavaScript execution. To simply extract all revealed passwords from the UI elements, the *innerText* property of the *root* element `document.getElementById('root').innerText` can be read.

6. Once the modification is done, simply repack the application `rm app.asar; npx asar pack ./app_unpacked app.asar`
7. Start *1Password.app* and observe the alert popping up.

As it was recommended in the previous ticket, 1Password should put a stronger emphasis on making it possible to install the application into a safe place. This usually requires the user to confirm the *User Account Control* dialog and disrupts the user experience of installation, but otherwise the trivial backdooring remains possible.

1PW-18-006 WP2: Malware can trivially intercept URL handler *(Medium)*

***Note**: As noted in the recommendation of this finding, this issue is hard to resolve. On Windows and Linux, limitations exist in terms of how much custom applications URIs - and many other means of inter-process communication - can be trusted when assuming a compromised machine.*

The 1Password Security Design white-paper notes that there are limited protections for certain information - including the secret key - on compromised machines. This issue unfortunately highlights a security limitation of these platforms in that same category. 1Password can't fix this issue right now, but if anything changes on Windows or Linux desktops that allows the developers to defend against this in the future they will make use of it.

It is trivial for Windows or Linux malware running as the same user as 1Password to modify the *onepassword://* URL handler in the registry. This allows the malware to intercept each URL handler invocation, log the data and forward it to the real 1Password binary. For the user there is no noticeable difference that this is happening.

PoC for Windows:

The following code is a simple C# application that prints the received arguments and then passes the arguments to 1Password. This application should be compiled and named *1Password.exe*.

```
using System.Diagnostics;

Console.WriteLine("Intercepting onepassword://");
if (args != null && args.Length > 1)
{
    Console.WriteLine("Received program arguments:");
    for (int i = 0; i<args.Length; i++)
        Console.WriteLine(" | "+args[i]);
    Console.WriteLine("Redirect to 1Password");
```

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

```
    Process.Start("C:\\Users\\visua\\AppData\\Local\\1Password\\app\\
8\\1Password.exe", args[0] + " " + args[1]);
}
Console.ReadKey();
```

Steps to reproduce on Windows:

1. Find the *onepassword* entry in *HKEY_USERS\...\Classes\onepassword*
2. Go to *…\onepassword\shell\open* and modify the *1Password.exe* path to the PoC's interception application from above.
3. Open any *onepassword://* URL from the browser and observe how the PoC application dumps the forwarded data
4. *onepassword://team-account/add?email=mrrooni%401password.com&key=A3-46REFP-XBLVZ5-YB5DZ-SPS9L-XFEFG-T3SV3&server=https://dark-mode.b5dev.com*

Steps to reproduce on Linux:

1. Copy */usr/share/applications/1password.desktop* to a new file called *fake1Password.desktop*
2. Modify the file to execute a program that should intercept the URL handler.
3. Install the new URL handler definition with *xdg-desktop-menu install fake1Password.desktop --novendor*
4. Open a *onepassword://* URI.

It is a very difficult task to protect 1Password from malware running on the same machine. Registering the same URL handler is a particularly easy method for malware to leak credentials from 1Password. Note that the Android equivalent of this attack is called *"insecure activity start*".

There seems to be no easy way to fix this because Windows does not offer any way to tie registry permissions to process signatures. Thus, any program running as the same user has access to the same registry. It probably would require a much more complex program architecture, wherein a privileged 1Password service is running in parallel to being responsible for management of the registry entries. The increased complexity would also add the risk of privilege escalation exploits.

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.

1PW-18-001 WP1: Insufficient validation of emails in *onepassword* URL *(Info)*

***Note**: The 1Password team has accepted this finding as a best practice issue and are looking to have this addressed in a future version of the app. That version will use improved validation of email addresses.*

It was found that the application does not properly validate contents of the email parameter received via *onepassword://* URLs. While other parts - such as the *server* value - are correctly validated, the *email* parameter allows any characters, for example *"://","<>","{}","[]"*. Additionally, the domain name part can also be empty. This unnecessarily increases its usability for other exploitation scenarios. However, due to the fact that the application properly escapes content added to the sign-in form, the behavior can currently be used for some Phishing and similar approaches exclusively, as shown below.

Example URL:
```
onepassword://team-account/add?email=if+not+work+visit+https://
evil.com+for+login@&key=A3-P76RPV-6MS29W-569CZ-PGMR2-DJHHY-XXXXX&server=https%3A
%2F%2Fseba-1pw18.b5test.com%2F
```

While this does not introduce a security issue itself, it is nevertheless recommended to introduce a proper input-validation that only permits characters that are allowed during account-creation.

1PW-18-002 WP1: Missing deprovisioning of TPM enclave key *(Low)*

***Note**: The 1Password team has accepted this finding as a best practice issue and has addressed it. Cure53 tested an early version of this feature, but publicly available versions of the app with this feature are not affected. Released versions of the app will deprovision the TPM secrets upon disabling the associated feature or when users uninstall the app.*

During the assessment of the newly introduced Windows Hello feature, the discovery was made that the enclave key remains in the *Microsoft Passport Key Storage* after the application has been closed or uninstalled. Since Windows binds access to those keys on the application-level and generally prevents exporting the private key, the risk can be considered rather low.

Show stored keys:
```
certutil -csp "Microsoft Passport Key Storage Provider" -key
```

Remaining key material:
```
S-1-5-21-2039661907-1149013884-2761926641-1001/336fcb5c-76ce-46f4-b491-
84fdabd1729b/S-1-5-21-2039661907-1149013884-2761926641-1001//1Password-Enclave-
Key
```

Nevertheless, in order to further harden the application against potential local malware threats, it is recommended to also remove the key material from the key storage after the 1Password application was closed or at least deinstalled. This can be done via the *DeleteAsync*[1] function of the *KeyCredentialManager* API.

1PW-18-004 WP2: DoS for update via Windows registry deletion *(Low)*

Note*: The 1Password team has accepted this finding as a low priority issue and is looking to have this addressed in a future version of the app. That version will not fail if the Windows registry doesn't contain the expected information.*

The Windows registry is generally an important threat surface for 1Password because even when installed in a trusted location, other apps could modify the values. It was found that when the registry entry is deleted, the 1Password application silently fails to check for updates. Visually there are no indications that it failed; the user simply sees it as if the latest version was being used.

Steps to reproduce:
1. Use the registry editor or programmatically delete the complete entry *\HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\1Password*
2. Restart 1Password and click on "*Check for Updates…*"
3. The settings screen will then display *"Version 8.5.0"* with no way to actually check for an update.

1Password has to be very careful using the Windows registry as a trusted source for data, as it can easily be manipulated by malware on the same machine. This threat surface will still exist once 1Password is installed in a protected folder, thus it has to be used with care; see also 1PW-18-006 for additional comments.

[1] https://docs.microsoft.com/en-us/uwp/api/windows.security.credentials.keycre...ger.deleteasync

1PW-18-005 WP1: Outdated Electron version used (*Info*)

***Note**: The developers have investigated this finding and have confirmed this is an artifact of the testing process. As a result they have not accepted this issue. At the time Cure53's testing started, the provided build indeed used a slightly older version of Electron. However, there are internal processes in place to update to the latest version of Electron in a timely fashion. 1Password updated to a version of Electron unaffected by the issues mentioned in this finding on December 1st, 2021, soon after Cure53 started their test.*

During the assessment of the provided sources, the discovery was made that the 1Password Core application is linked to an outdated version of Electron.

At the time of writing, the latest available version in 15 is *15.3.2*[2] and the linked version is *15.2.0*. In version *15.3.1*[3] some security updates for the embedded Chromium browser were added. They address some heap buffer overflows and 'use-after-free' problems in Skia, V8 engine and other components. Under certain conditions, this might allow an adversary to escape the sandbox via a crafted HTML page.

However, the resulting impact can be considered rather low due to the fact that an attacker needs to be able to render own HTML within the 1Password applications. Nevertheless, it is recommended to always keep the version of Electron up to date. If an update is not immediately feasible, it is recommended to at least ensure the patch notes do not contain any security-relevant fixes.

[2] https://www.electronjs.org/releases/stable?version=15
[3] https://www.electronjs.org/releases/stable?version=15#other-changes-1531

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

Conclusions

In this round of audit, Cure53 was tasked with reviewing the newest, next iteration of 1Password Core apps, with a particular emphasis on new features for Windows and Linux. To clarify, these features were the Windows Hello integration, the improved browser integration on Linux and the CLI integration on Linux. While a special focus was given to the aforementioned areas, other parts of the applications were also examined by Cure53 in considerable depth.

To reiterate the context and resources, three members of the Cure53 team completed the project over the course of fourteen days in November and December 2021. While the overall number of six issues were found during the audit, two of them are exploitable and four can be considered as a hardening recommendation and best practice advice.

The report separates issues into two Work Packages. The first package (WP1) relates to problems found in the special focus areas, while the second work package (WP2) shows issues that can affect 1Password Core applications if targeted systems are vulnerable via local malware.

Cure53 needs to underline that the provided builds generally share the same codebase and differ mainly in the parts that have to do with providing the applications' functionality on the respective operating systems. This also made it rather easy to compare different functionalities between the operating systems, fostering understanding why and how certain types of issues can be applicable to the other operating systems as well.

In the first step, Cure53 reanalyzed the configuration of the Electron framework, so as to see if any new changes had been made. All the important flags, such as *nodeIntegration* and *contextIsolation*, are set accordingly, thus reducing the impact of post-exploitation. However, the current used version lacks important patches that address some buffer overflows and 'use after free' flaws in the embedded Chromium browser (see 1PW-18-005). It is, therefore, recommended to ensure that an update to the latest version takes effect.

An additional task added to the test encompassed diving deeper into the IPC mechanism of 1Password on Linux. Even though it was an extra task, it fit well into the main focus area of this round and did not lead to a narrower coverage for the original areas. 1Password is planning to simplify the authentication of the IPC on Linux by checking the effective group ID of the connected peer. This implementation was reviewed and tested, especially with regards to user-namespaces and whether they could fool the check. No issues could be found in this realm.

One of the main focus items on Cure53's priority-list was the newly implemented Windows Hello feature and the corresponding TPM usage. A TPM is generally used by Windows for creating and storing cryptographic keys. It also can offer system measurements for verifying system's integrity in terms of secure usage of a TPM. Most of the TPM functionality is provisioned by Windows itself, whereby access is generally handled via corresponding Windows APIs, such as *TPM Base Services (TBSI)*[4] or the *KeyCredentialManager* API. Thus, the 1Password application makes proper use of those functions during key derivation. In particular, reliance on *RequestCreateAsync* or *OpenAsync* of the *KeyCredentialManager* API signifies that implementation standards proposed by Microsoft are followed in the Windows Hello integration by 1Password.

The sources revealed that 1Password uses a static credential name. However, a further test revealed that Windows separates access to those keys at the application-level. It was neither possible to access credentials created by another process. nor to create a fake credential entry with that static name. This is because Windows generally separates such entries via SIDs and other UUIDs, doing so at the application-level. Additionally, measures are correctly used to properly verify the system's integrity. Particularly, PCR banks are checked for zero entries which could miss a compromise of the integrity of the system.

Only one issue was found by Cure53 in the area above and is related to a missing deprovisioning of the created key entry. However, due to the fact that private keys generally cannot be extracted from the TPM, this risk can be considered low and was only added as a hardening recommendation to prevent abuse in potential post-exploitation scenarios (see 1PW-18-002).

Generally, the Windows Hello feature follows a new approach of trusting key material created in the TPM. While this makes perfect sense, it also introduces some new risks in other areas of Windows Hello. For example, a vulnerability to bypass Windows Hello login via face recognition feature[5] was found. While Windows generally trusts USB devices during authentication for using webcams, the attack surface remains. Although Windows fixed the found issue, this shows that 1Password could be affected in the future if another Windows Hello bypass emerges. Nevertheless, Cure53 sees the likelihood of being exploited as low, due to the fact that Windows might fix these types of vulnerabilities in a shorter time.

Cure53 also examined the slightly improved changes related to the Browser Integration feature on Linux and the corresponding keyring helper handling, which did not lead to

[4] https://docs.microsoft.com/en-us/windows/win32/api/tbs/
[5] https://www.cyberark.com/resources/threat-research-blog/bypassing-windows-hello...lastic-surgery

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

any finding in this area. Next, Cure53 also reviewed the 1Password Core applications in terms of security against several local attack scenarios.

During a previous round it was found that malware on macOS could trivially backdoor the *html/js* source files, which allows very simple access to the data stored in 1Password. As this round had a focus on Windows, the issue resurfaced in this OS. 1Password is currently installed into *AppData* which allows basic malware to easily modify and backdoor the app (see 1PW-18-003). However, it was already announced that 1Password might be installed into the *Program Files*, which would protect against these threats.

Another general threat surface on Windows that has not been clearly documented before spans non-administrative malware which can easily modify the registry entries used by 1Password. A small Denial-of-Service issue was found when the entry was deleted: the application fails to check for updates and shows no indication of that issue to the user (see 1PW-18-004).

This also leads to another general attack technique that malware on Windows and Linux could succeed with. For Windows and Linux, it is easy for unprivileged malware to modify the URL handlers, as demonstrated in 1PW-18-006. This can be used to easily redirect and "intercept" *onepassword://* URLs. These URLs carry some important data, but do not allow for a full vault takeover due to the missing password.

When looking at the reported issues with moderate severities, they are all in the realm of a threat model that 1Password cannot really protect against anyway. Yet, it is still commendable that 1Password has an interest in tackling this threat surface. The efforts can significantly lower the risk of very simple malware attacking 1Password users.

It is generally a good design to have no privileged helper component as part of 1Password. This approach eradicates the attack surface for privilege escalations. However, a middle-ground of using a privileged installer, which can manage locking down Windows registry keys, installing the application into a safe location and so forth, is a good step. On Linux this is already the case and prevents easy backdooring from malware, which has proven possible on Windows.

In addition, the examined codebase of the corresponding 1Password Core applications seem solid in regard to the security posture. It adheres to common best practices, which results in this very good state.
In particular, it was checked whether the application makes use of dangerous functions or implemented common pitfalls that could lead to major vulnerabilities, such as

injection-based attacks, insufficient escaping of user input, path traversal issues, remote code executions or privilege escalations attacks. None such flaws were spotted.

All in all, the examined 1Password Core applications for Windows, Linux and macOS and the related specific areas leave a very good impression in terms of security. Cure53 more broadly confirmed during this audit that the provided builds have the capacity to fend off many different attacks. This clearly shows that the 1Password team is aware of the problems that modern applications tend to face. As a result, only one *High* and one *Medium*-scored exploitable issues related to local malware threats could be found in this test.

The very good result from this audit also clearly shows that 1Password is versed in using and implementing existing security measures as much as possible. This helps accordingly protect the applications against a range of different threats. However, some weaknesses were found, mostly affecting the Windows client. They basically allowed a trivial injection of malicious HTML/JavaScript into the context of the running application. Additionally, some new attack vectors were discovered and show how registry entries can also be used to attack the application. Still, well-chosen languages, such as Rust in combination with a well-designed architecture complex underlines the solid picture Cure53 acquired during the audit. Assuming that all relevant issues get fixed, Cure53 can evaluate 1Password Core applications as properly secured for a continued production use. The complex is capable of delivering a secure foundation.

Cure53 would like to thank Stephen Haywood, Adam Caudill and Rick van Galen from the 1Password team for their excellent project coordination, support and assistance, both before and during this assignment.