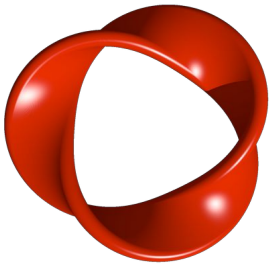

Secfault Security

1Password for Mac

Security Assessment

Report

for

Agilebits Inc dba 1Password

4711 Yonge St., 10th Floor
Toronto,ON M2N 6K8 AgileBits

- hereafter called "Agilebits" -

Document History

Version	Author	Date	Comment
0.1	Jennifer Gehrke	2022-11-22	First Draft
0.2	Gregor Kopf	2022-11-30	Additions
0.3	Dirk Breiden	2022-12-02	Internal Review
0.4	Gregor Kopf	2023-03-01	Added Customer Feedback
1.0	Gregor Kopf	2023-03-03	Final Version
1.1	Gregor Kopf	2023-03-17	Additional Grammar and Layout Corrections

Table of Contents

1 Executive Summary

Secfault Security was tasked by Agilebits with a security review of selected components of the 1Password ecosystem, namely version eight of the macOS application. The review has been performed in the time frame from 2022-11-21 to 2022-12-02. This document describes the results of the project.

During the review a number of issues, which are described in detail in section 4 of this document, have been identified. The more severe issues include insecure file ownership handling in the updater described in section 4.5. Further, a possibly weak default behavior was identified in the context of the optional Multi Factor Authentication (MFA) (please refer to finding 4.3 for more details).

Section 5 of this document provides a number of additional observations and recommendations for further strengthening the security aspects of the solution. Those relate to the clear mapping of items to the correct vaults in the client UI as well as different defects in the context of CSV handling (please also note the security issue presented in section 4.2).

Overall, the reviewed client implementation left a positive impression. Only a few issues could be identified that originate from bad coding patterns. This indicates that the client has been implemented with security best practices in mind.

After having received a draft version of this document, Agilebits provided feedback on the identified issues, which can be found in section 6 of this document. Secfault Security would like to highlight that with respect to the findings 4.3 and 4.6, a number of implicit assumptions have been made, which do not fully correlate with the design goals of Agilebits. In order to provide full transparency, the respective findings have not been edited. Please refer to section 6.3 and 6.6, where an explanation of the MFA design is provided.

2 Overview

1Password is a password manager product developed and maintained by AgileBits Inc. The solution provides a secure place for customers to store various passwords, software licenses, and other sensitive information in virtual vaults. Agilebits tasked Secfault Security with a review of the new macOS version of the 1Password software.

In section 2.1 of this document, a description of the project's scope is provided. Section 2.2 provides details on the test procedures.

2.1 Target Scope

The below list of aspects summarizes the scope document provided by Agilebits[1]. The audit focused on the client implementation as utilized by 1Password for Mac.

2.1.1 Critical User Flows

- Master password and secret key based login
- Master password and secret key based login with activated MFA
- Unlocking of Vaults

2.1.2 MacOS Specific Features

- Removal of main process from Apple sandbox
- Biometric unlock based on TouchID, Apple Watch, or hardware security keys
- New PKG installer

2.1.3 1Password 8 Added Features

- Data Import
- Secure file attachments
- Move item / share items
- Family / Shared Vaults + New sharing details
- Travel Mode
- Password history
- Item archiving and deletion features

The MDM-related aspects of the new PKG installer could not be covered during the assessment, due to time constraints in the upfront agreed time frame.

1 Q4-22_1Password_8_for_Mac.pdf

2.2 Test Procedures

The overall project followed a white-box approach, which means that Agilebits provided the source code, the compiled binaries and technical documentation for the solution. Therefore, the solution has been analyzed by performing a source code review, as well as targeted dynamic testing.

The source code review has been performed in a manual fashion, i.e., without relying on automated vulnerability scanners or similar tools. Besides identifying possible classical implementation weaknesses, one main focus of the review was the identification of potential logic problems. This requires an in-depth understanding of the solution's inner workings, which is best achieved by a manual process.

The dynamic tests have been performed in a targeted fashion. On the one hand, this served the purpose of validating issues identified during the source code review. On the other hand, dynamic tests were also performed to obtain a better understanding of the overall solution and the interplay of its individual components.

2.3 Project Execution

The project has been executed in the time frame from 2022-11-21 to 2022-12-02.

The consultants assigned to this projects were:

- Jennifer Gehrke
- Finn Westendorf
- Gregor Kopf

3 Result Overview

An overview of the project results is provided in the following table.

Description	Chapter	Type	Severity
Unprotected Item Fields in 1PIF2 Format	4.1	Code	Medium
CSV Injection In Export	4.2	Code	Low-Medium
Questionable Security Advantage of DSecret MFA	4.3	Design	High
Master Password and Secret Key in Export Files	4.4	Design	Medium
Unsafe chown Call	4.5	Code	High
Insufficiently Authorized MFA Management	4.6	Code	Medium
Unsafe Client Storage of Secret Key	4.7	Design	Medium
Import of References to Existing Vaults	4.8	Code	Low
Vault Permission Preview Incorrect	4.9	Observation	N/A

Each identified issue is briefly described by its title, its type, its exploitability and by the impact of a successful exploitation. Technical details for the individual issues are provided in the respective sections of chapter 4 of this document. Details regarding the vulnerability rating scheme used in this document are provided in section 7.

4 Results

The issues identified during the project are described in detail in the following sections. For each finding, there is a technical description, recommended actions and - if necessary and possible - reproduction steps. For details regarding the used vulnerability rating scheme, please refer to section 7 of this document.

4.1 Unprotected Item Fields in 1PIF2 Format

Summary

Type	Location	Severity
Code	Data Export Format	Medium

Technical Description

As part of the audit of the data import feature, a static code review was performed. Parsers for four different formats were detected: CSV, 1PUX, 1PEX and 1PIF2. The last two formats support the protection of the exported contents with AES-GCM encryption, which provides data integrity protection, due to its AEAD nature. For the 1PIF2 format, this encryption was found to be applied incompletely on the final vault item, whereby the respective fields could be manipulated.

Discussions with Agilebits and dynamic tests revealed that this format is not actively used by the provided client. The issue is nevertheless raised since it relates to the issue "4.2.2 Unauthenticated Meta Information in Database" reported in the SSH agent security audit in March 2022 and should be addressed in case the format is ever supported for end users.

The routines parsing the different vault items from the 1PIF2 file were found to replicate certain fields without previous decryption. The respective implementation is located in the file `op-import-export/src/models/translation/onepux/v2.rs`:

```
1479     pub(crate) fn decrypt_1pif2_vault_item<R: Read>(
1480         item_data: R,
1481         keys: &Keys,
1482     ) -> Result<InnerVaultItem, error::Error> {
1483         let e_vault_item: EncryptedVaultItem =
1484             de_from_reader(item_data,
DeserializeError::EncryptedVaultItem)?;
1485
1486         let overview = {
1487             let overview_bytes =
decrypt_message(&e_vault_item.encrypted_overview, keys)?;
1488             de_from_slice(&overview_bytes,
error::DeserializeError::ItemOverview)?
```

```
1489         };
1490
1491         let details = {
1492             let details_bytes =
decrypt_message(&e_vault_item.encrypted_details, keys)?;
1493             deserialize_item_details(&details_bytes)?
1494         };
1495
1496         Ok(InnerVaultItem {
1497             uuid: ExportItemUuid::from_string(e_vault_item.uuid),
1498             fav_index: e_vault_item.fav_index,
1499             created_at: e_vault_item.created_at,
1500             updated_at: e_vault_item.updated_at,
1501             trashed: e_vault_item.trashed,
1502             category_uuid: e_vault_item.category_uuid,
1503             overview,
1504             details,
1505         })
1506     }
```

Here the encrypted vault item is first read from its JSON representation, decrypting the `encrypted_overview` and `encrypted_details` fields afterwards. Finally an `InnerVaultItem` is created from this information, copying the values of the `uuid`, `fav_index`, `updated_at`, `trashed` and `category_uuid` directly from the encrypted item.

Example 1PIF2 files contained in the client source code repository confirmed this impression. By unzipping the file `op-import-export/test_exports/1pif2/valid/base.zip`, a directory `<account_uuid>.account` containing `<vault_uuid>.vault` directories is created. Those contain the encrypted items in files named `<item_uuid>.enc_item`, with the following exemplary content:

```
{
  "encryptedOverview": {
    "iv": "432W2D5yECNbjRNV",
    "data": "BqhydtVAQ7MOGto0fmlPLSP9EAeLbsI_N5OF-
IyyQQ0fdi1AQqAXy4H09_Si9da5aE0A0mCy_A-
QG31XVFFPPHQX2iyKIZ7FLnbdQgqYM5mgJzTObX7uv0cxs3iVojKCeYigGBmTEDsyuYy1IOrISsooLJ
tXERt1GmpD-
btUUjDaV3xTIJlbF2mHio11PJthlCqeJlN5wMmtdjLcanxyVtK6c4IWLH6wJ0ypvTtYJUAfH2QgrblK
ddwid8kECw2GPwA1",
    "enc": "A256GCM",
    "kid": "ubhdkwxzsjanjdz5mmuzpabew4",
    "cty": "b5+jwk+json"
  },
  "categoryUuid": "001",
  "encryptedDetails": {
    "iv": "0BpN2HEm_DxPFOT2",
    "data": "BILmo3iGBE9ZOkgJyC-
JxdTdb2nGWZN92rAemN6NTzalgkdUe8QisUVAxzEZK0x_lBYzpVHUI0VEjw0QBOvuHAr9oFBpba94_V
```

```
NhCCRSqeZ1TTfwGqViq3yrDl5vXfMx-axhicFsXjlC0p-
FtkSb4qRxWwOj8fsamWESt75fRaSc9hE1VrJhwPdXRT-
Jd8h8ZksYphHjaOx9jc2_jsvnNcHmVpCLfqhJ8f1v-xSAonYItqkVe03ajW_ztfkUd83WFDxkQPs",
    "enc": "A256GCM",
    "kid": "ubhdkwxzsjanjdz5mmuzpabew4",
    "cty": "b5+jwk+json"
  },
  "uuid": "se3rseo6kvavxmca7jm2hjhndy",
  "favIndex": 0,
  "createdAt": 1247253633,
  "updatedAt": 1540244860,
  "trashed": true
}
```


As one can easily observe, the listed item fields are contained in plain text. Since the source code review showed no additional integrity protection measures, an attacker should be able to manipulate these fields. For considerations regarding the sensitivity of the `favIndex` field please refer to section 5.1.

Recommended Action

It is recommended to include the whole vault item representation in the applied AEAD encryption to protect the named fields. Further, it is advisable to consider whether the design for encrypted formats should be adjusted in a way that entries of the exported archive cannot be removed or permuted unnoticed. The current implementation iterates through the archive entries based on the available file names matching an extension, e.g. `.enc_item`, and decrypts each single file with the available key material. Instead, information on the vaults that exist in an account and items that belong to an exported vault could be integrated in the account or vault serialization. This information, for example cryptographic file content hashes, should allow to detect the removal or exchange of file contents.

4.2 CSV Injection In Export

Summary

Type	Location	Severity
Code	CSV Export	Low-Medium

Technical Description

Data like saved logins may be exported as `.csv` file, if the correct permissions are present. Most software that displays CSV files can use so called formulas, which can be used to perform various tasks, including potentially executing code. It was found that the 1Password export does not escape the characters needed to create formulas, which results in a vulnerability known as "CSV injection".

When a malicious user can edit an item, other users exporting this item and looking at the resulting CSV e.g., in Excel, might be compromised.

Recommended Action

Secfault Security recommends to follow the approach outlined by OWASP[2] to escape the cell content correctly:

- Wrap each cell field in double quotes
- Prepend each cell field with a single quote
- Escape every double quote using an additional double quote

Alternatively existing logic can be reused here, as a CSV injection bug was fixed by 1Password in a previous assessment of a different software component[3].

Reproduction Steps

To reproduce this issue, please enter a known CSV injection payload in a field which will be exported e.g., as a password, and create a CSV export.

- First please open the macOS client, login and make sure at least one Login exists
- Please modify the existing login, so that the password is `=cmd|' /C calc'!A0`
- Now open the export dialog `File > Export > Name Of Organization`
- Please enter your password
- Now select CSV as export type and save the file

When the resulting `.csv` is opened in a text editor, there should be a cell with the content `=cmd|...`

2 https://owasp.org/www-community/attacks/CSV_Injection
3 https://bucket.agilebits.com/security/378.2101-Recurity_Labs-Report-B5-v1.0.pdf

as set before. This means CSV formula injection was successful.

4.3 Questionable Security Advantage of DSecret MFA

Summary

Type	Location	Severity
Design	DSecret MFA	High

Technical Description

During the source code review of the client login routines, several MFA mechanisms were found to be supported to protect the account in cases were attackers gained access to its master password. One of these is based on so-called DSecrets, which are device-specific secrets that were found to be stored in the OS keyring or in the local storage of the browser. Those are enrolled to the device on the first successful MFA via one of the other enabled mechanism, e.g. TOTP. This way, a user can still establish connections to the backend by unlocking the vault with the master password alone or by using biometric unlock. While the idea behind this characteristic is plausible from the usability perspective, it is considered highly questionable that it is activated by default and the DSecret validity is in principle endless, according to statements of Agilebits.

The result of this feature is, that solely the device enrollment can be considered to require MFA. However, the enrollment already requires two secrets: the master password and the account's secret key. While both are technically of the same type, they can still be considered to have a different risk of being compromised. While the master password is regularly used, the secret key is utilized during client enrollments only, whereby it is significantly less exposed. Consequently, only a small security advantage is gained by the current MFA implementation: This depends on the protection the specific client can apply to its storage. In case a OS keychain is available and used, the ability to execute code on the specific device functions as a second authentication factor. The more comparable the security of the DSecret storage gets to that of the secret key (please note that section 4.7 recommends the storage of the secret key in the keychain), the smaller the advantage will be. In general, users, especially in business contexts, would rather expect that the MFA has to be performed at least periodically.

Recommended Action

Secfault Security recommends to introduce configuration options for the utilization of the DSecret mechanism. Those should cover both the ability to fully deactivate this mechanism for an account as well as the option to specify a expiry time-frame. This should be possible either by the user directly or by the superordinate business account. Changes to these settings must be authorized the same way as turning off MFA (see the recommendations given in section 4.6).

On enabling MFA for an account or all team members, the user should be requested to confirm that this mechanism is used and must specify a time-frame for which DSecets may be accepted by the

server. Default values aligning with the security needs of the respective account type should be preselected, e.g. the DSecret expiry after 24 hours for personal accounts and considerably shorter time frames for business related accounts. The user should be informed that the security advantage of the activated MFA highly correlates to the configured duration.

Reproduction Steps

Activate MFA for an account of an arbitrary type and setup a new client for it. Note that MFA via TOTP or a security key is required to setup the client. Now lock the client or reboot the device and observe that MFA is no longer required for unlocking 1Password including a connection to the backend afterwards.

For the macOS client, the mentioned DSecret can be found in the `login` keyring. Note the time of registering the new client and search for an entry modified at this time. The entry name should be prefixed with `1Password:dsecret-` and the item does not expire.

4.4 Master Password and Secret Key in Export Files

Summary

Type	Location	Severity
Design	1Password for Mac	Medium

Technical Description

To get a general understanding of the different import formats supported by the macOS client, export files were generated for the used test accounts. While reviewing their contents it was found that some export files contain the account's master passwords and for `1pux` files also its secret key. This behavior seems to be caused by the default items generated by 1Password for each new personal or family account. Those are stored in a vault named "Personal" or "Private" and contain among others a login item named "1Password Account (<user/family name>)". This stores the account's master password and secret key:

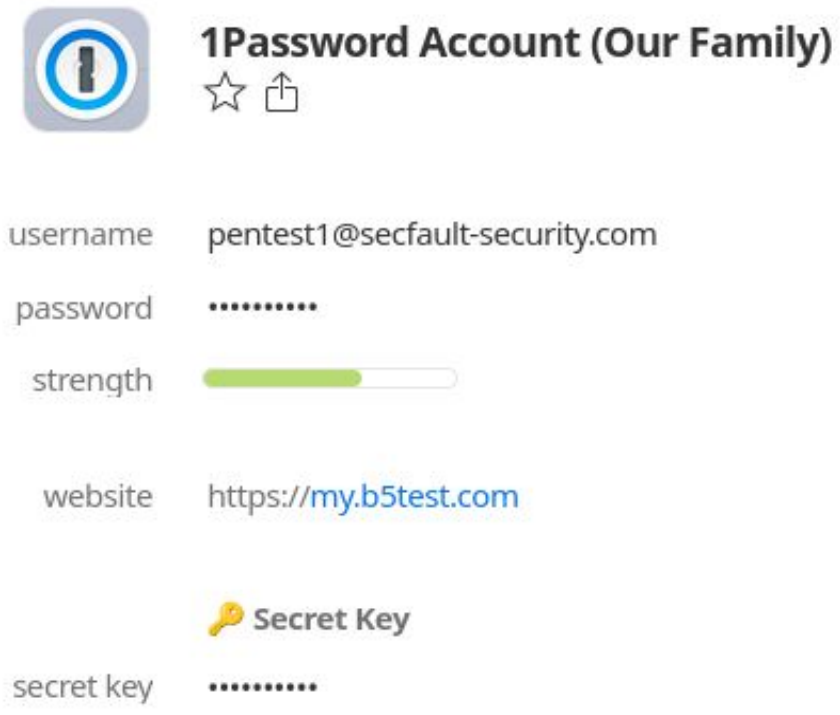

Figure 1 - 1Password Account Login Item

This behavior might pose a security risk, since an attacker with access to an export file (currently solely plain text formats are supported) will not only gain access to the stored content, but also to

future items stored or shared to the account by having the ability to fully hijack it.

Recommended Action

It should be considered to remove this item from exports by default. It could be included on demand of the user with a appropriate security clarification.

Reproduction Steps

In order to reproduce this issue, please create a new personal or family account and login for the first time via the Web UI or one of the offered clients. Note the existence of a "Personal" or "Private" vault and the "1Password Account (<user/family name>)" login item that is stored in it. Its detail view should confirm the availability of the master password and secret key as shown in Figure 1.

4.5 Unsafe chown Call

Summary

Type	Location	Severity
Code	apple/macOS/FileManager+Authorization.swift	High

Technical Description

While reviewing the new installer implementation, it was found that the 1Password Updater, which is installed alongside the main application, exposes a weakness that allows local attackers to gain elevated privileges by exploiting an insecure call to the `chown` binary.

The following excerpt from `FileManager+Authorization.swift` illustrates the problem:

```
   private func moveToPathWithForcedAuthorization(at source: URL, to
destination: URL) -> Bool {
      let src = source.path
      let dst = destination.path

      let tmp =
destination.deletingPathExtension().appendingPathExtension("old").appendingPath
Extension(destination.pathExtension).path

      var sb = stat()
      if stat(src.seriallizeToC(), &sb) != 0 || stat(tmp.seriallizeToC(), &sb)
== 0 {
         return false
      }

      setenvWithString("SRC_PATH", src, 1)
      setenvWithString("DST_PATH", dst, 1)
      setenvWithString("TMP_PATH", tmp, 1)

      var shellCommand: String?

      if FileManager.default.fileExists(atPath: dst) {
         shellCommand = String(format: """
                                             /bin/rm -rf \"$DST_PATH\" &&
                                             /bin/mv -
f \"$SRC_PATH\" \"$DST_PATH\" &&
                                             /usr/sbin/chown -R %d:
%d \"$DST_PATH\"
                               """,
```

```
                                sb.st_uid, sb.st_gid)
[...]
      if let shellCommand = shellCommand {
         return runShellWithAuthorization("1Password needs to update some of
its files, which requires the password you use to log in to your Mac.",
shellCommand)
      }
```

It can be observed that the code first obtains information on the current file owner of the `source` directory, and then - after invoking `mv` - recursively changes the ownership of the destination directory to match the `source` file owner. This operation is performed with elevated privileges, as indicated by the `runShellWithAuthorization` call.

This is problematic insofar as the `source` directory could contain a hard link to a file that is owned by another user, such as `/etc/sudoers` that is owned by `root`. Changing ownership on such a link would result in the file becoming writable by less privileged users.

The code path is triggered by the 1Password Updater implementation, which has been confirmed by performing targeted dynamic tests.

Recommended Action

Secfault Security recommends to generally reconsider the shell-based implementation of the updater routine. This is due to the fact that - based on prior experience - such implementations are often hard to defend against various classes of attacks, including the described link-based attack, but also other attack classes such as race conditions.

If this is not possible or not desired, it is recommended to at least not use `chown` for explicitly changing file ownership. Rather, existing file permissions should be preserved by instructing the respective utility to do so. In case of `mv`, this behavior should already be implicit.

Reproduction Steps

In order to reproduce this issue, please proceed as described below.

In a first step, please create a root-owned file `/etc/sudoers.test`, e.g., by `sudo cp /etc/sudoers /etc/sudoers.test`.

Then please create a directory named `1Password.app` and link `/etc/sudoers.test` to this directory: `mkdir ~/1Password.app ; cd ~/1Password.app ; ln /etc/sudoers.test ..`

Now please invoke the 1Password Updater as follows:

```
env path=~/1Password.app /Applications/1Password.app/Contents/XPCServices/OP
Updater Service.xpc/Contents/Helpers/1Password
Updater.app/Contents/MacOS/1Password Updater.
```

An authorization prompt should be shown. After confirming it, please observe that

`/etc/sudoers.test` is now owned by a regular, non-root user.

4.6 Insufficiently Authorized MFA Management

Summary

Type	Location	Severity
Code	Web UI	Medium

Technical Description

One of the test-cases for MFA management involved activating and deactivating MFA for some of the available accounts. This functionality is solely available in the 1Password Web UI, where a violation of security best practices was identified for disabling MFA.

The basic idea of optional MFA mechanism is addressing cases where users want to protect against attackers who have gained access to the mandatory authentication factor. In the present setting, an attacker with this knowledge and with access to an authenticated Web UI session can disable MFA for the whole account and thereby any client. Note the security issue documented in section 4.3 that enlarges the risk of getting access to an authenticated UI session provided that the password is already known.

Further dynamic tests showed, that not only the routines for disabling MFA are affected, but also the request handlers allowing to replace or add a second factor. This way an attacker could add an alternative second factor for the account or replace an existing one. This might pass unnoticed by the actual user, since on a login based on DSecrets no MFA option overview will be shown, nor was an notification email received on replacing a TOTP factor.

A similar situation as on disabling MFA was observed when changing the account's email address. For this, the user needs to enter the code sent for verification to the new email address together with the master password. This is of relevance in the MFA context for accounts using MFA via codes sent to the account's email address.

Recommended Action

Since disabling MFA must be regarded as a critical account management option, it should be considered to authorize this action by all available authentication factors. At least, the deactivation of MFA should require the proof of access to a valid second factor.

The same measures should be applied, whenever a second factor should be replaced, including changes to the account's email address, or added.

Reproduction Steps

In order to reproduce this issue, please login to the Web UI using an arbitrary account that has MFA enabled, but is not forced to do so by a superordinate business account. Open the drop-down menu

offered next to the account name in the upper right corner of the landing page. Navigate to "My Profile" and select "More Actions -> Manage Two-Factor Authentication" in the left sidebar. Click on "Turn Off Two-Factor Authentication" and enter the account password in the shown overlay:

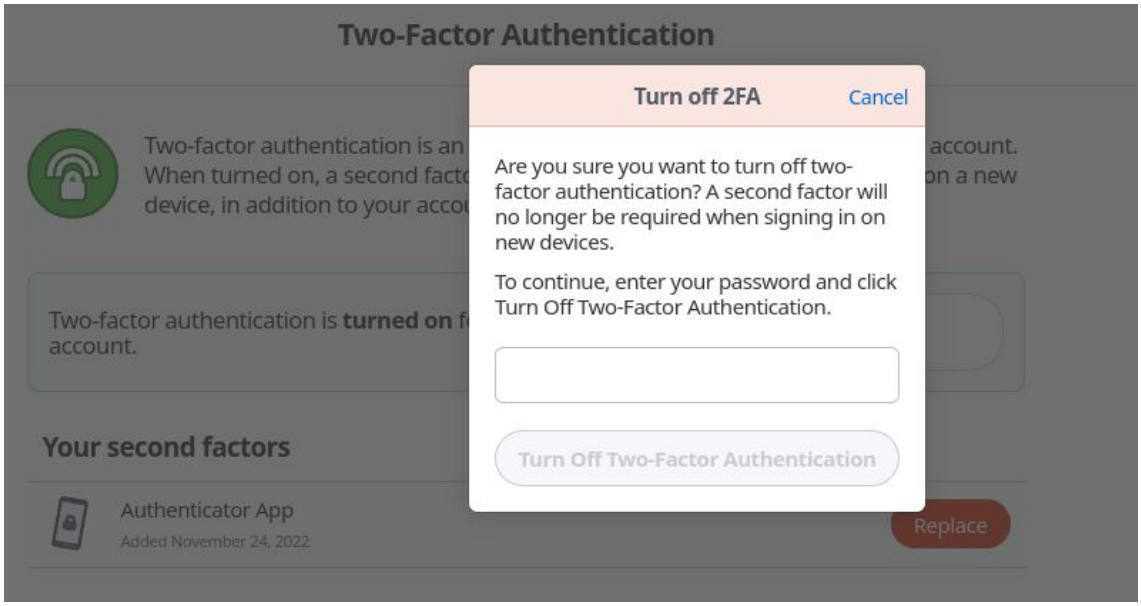

Figure 2 - Password Promp on Disabling MFA

On successful password verification, the page confirms that MFA was turned off without requiring any additional steps.

The replacement or creation of further second factors can be performed on the same Web page. Please note the missing authorization checks and notification email.

4.7 Unsafe Client Storage of Secret Key

Summary

Type	Location	Severity
Design	macOS Desktop App	Medium

Technical Description

To simplify the unlock process, the account's secret key is stored by the client after its initial submission and successful verification. Since the secret key is required to unlock the encrypted database contents and can therefore not be covered itself by this protection mechanism, the question arose how its storage is secured.

An inspection of the client's source code and database entries revealed, that the secret key is not cryptographically protected. It was found to be obfuscated by a procedure defined inside the file `op-crypto/src/secret_key.rs` that performs a bitwise xor with a constant string:

```
287     const OBFUSCATION_KEY: &[u8] = b"This is an obfuscation key used to
mask the secret key in the local database and nothing more. If this seems
interesting to you, come work with us :)";
288     pub(super) const OBFUSCATED_MARKER: &str = "obfus";
289
290     pub(super) fn obfuscate_key(secret_key: &SecretKey) -> String {
291         let str_form = secret_key.less_safe_human_readable_form();
292
293         let obfuscated: Vec<u8> = str_form
294             .bytes()
295             .zip(OBFUSCATION_KEY)
296             .map(|(c, k)| c ^ k)
297             .collect();
298
299         let mut encoded = op_encoding::hex(&obfuscated);
300         encoded.push_str(OBFUSCATED_MARKER);
301         encoded
302     }
```

The usage of this scheme was verified by inspecting the database contents. Those did contain entries with the expected `OBFUSCATED_MARKER` suffix.

Recommended Action

1Password for Mac already makes use of the OS keychain to store secrets in different contexts. Therefore it already supports a secured location for placing the account's secret key. Correspondingly, the management procedures of the secret keys should be adjusted to make use of this OS feature whenever available. This change should be applied to all client types, where a OS or

browser secure storage option might be applicable.

Reproduction Steps

In order to reproduce this issue, please setup a macOS client with at least one account that makes use of password authentication.

Search for the SQLite database and open it using the following command line:

```
$ find . -iname 1password.sqlite -exec sqlite3 {}\;
1|23|
{"account_state":"A","account_template_version":3407873,"account_type":"B","acc
ount_version":15,"base_attachment_url":"https://
f.b5test.com/","base_avatar_url":"https://a.b5test.com/","sign_in_provider":
{"type":"sk","secret_key":"155b4446705b417035436e5623543d464e522c5f24340d212e3c
18225e56347323370d2e234b2d67obfus","enc_unlock_key":
{"cty":"b5+jwk+json","kid":"system_lock_protector","enc":"A256GCM","iv":"TLVltQ
pK4dEx_-7Q","data":"6tVhlOeVEsf0PBX5lcwmtHxxvs-ZIzZV49f6GhN407QpvQL5Y6bwVrK6E-
ba_m-
YDI4kEWmBmpMzQQJFKESEMqTkd2IRrD4Hke1_yyGjrrRvjlX3MJ5he_87fMikBh48dBM7GieUdXCPdB
fmGrCwhY4HVXJUttd3YvaOtxZDmcRpxjIMtoGWAmIxTc99siaJm-jktxs"}},"enc_srp_x":
{"cty":"b5+jwk+json","kid":"srpxkey","enc":"A256GCM","iv":"k4P7D-
rvxByTopdd","data":"SNCkhOLbG4efd4EubQzH2Xq2Hv4YxQfeQPRN68l5B8i4SW_is33iShpli-
vQ2dPe"},"sign_in_url":"https://
secfaultsecuritygmbh.b5test.com/","team_avatar":"","team_name":"Secfault
Security
GmbH","updated_at":1669653353,"user_avatar":"","user_email":"pentest1@secfault-
security.com","user_name":"Jennifer","user_keyset_version":24,"user_uuid":"NUND
LBN5FJFZFFWS6LYXYR3FGI","user_version":3,"acl":786433,"device_uuid":"rs2ptxeh2q
moueyc5d6tbjbb2u","freeze_at":1669648883,"billing_status":"F","storage_capacity
":25600,"storage_used":1,"account_template_language":"en-
US","enc_local_validation_key":{"cty":"b5+jwk+json","kid":"core-setting-
authkey-wrapper","enc":"A256GCM","iv":"YXJDG2zXVMRQbRNw","data":"yzyia5aWptZ-
zGv_Rtk6K5tpz76ACOA7c8vWuxdK1lWUC3ynkRYM6Fj_1qdD6eHH"}}|
IH4BDH3TM5DKNOIGLVJDDLDOSM
```

The above JSON entry contains the secret key in the `secret_key` field of the `sign_in_provider` object. Note that the value of the `secret_key` field has the expected suffix `obfus`.

4.8 Import of References to Existing Vaults

Summary

Type	Location	Severity
Code	Data Import	Low

Technical Description

One of the most relevant security aspects in the context of the data import offered by the client, is the separation of vaults and items created during this with respect to already existing data. Since new vaults are created to store the imported data, it was investigated what options are offered to reference or link other items. In this context a new feature supported by the browser extension was identified, which allows to store information on the used login for web pages offering SSO authentication, e.g. using social media accounts. This is achieved by adding special fields to the login item that is otherwise only storing the Web page's URL.

In the JSON serialization of the resulting item in an 1PUX export one can see that both a vault UUID and item UUID for the respective SSO login can be specified:

```
"sections": [
  {
    "title": "Saved on passport.alibaba.com",
    "name": "Section_76rzqdvqk2nixi3yjgvimudm7u",
    "fields": [
      {
        "title": "single sign-on",
        "id": "ybjp5f3bpl5dmw4gkqzk6e3gkq",
        "value": {
          "ssoLogin": {
            "provider": "google",
            "item": {
              "vaultUuid": "2gk4fuezacl72ybikd3mti7cci",
              "itemUuid": "rpg2krm4yuzh3vczrttwkqiw5q"
            }
          }
        },
        "indexAtSource": 0,
        "guarded": false,
```

In the implementation of the respective deserialization routines in the file `op-import-export/src/models/translation/onepux/v2.rs` one can observe that the information is copied literally from the imported data:

```
855             SectionFieldValue::SsoLogin(attributes) => {
```

```
 856                 op_model::SectionFieldValue::SsoLogin(attributes.map(|
attributes| {
 857                     let item = attributes
 858                         .item
 859                         .map(|attributes| op_model::SsoLoginLinkedItem {
 860                             vault_uuid:
VaultUuid::from_string(attributes.vault_uuid),
 861                             item_uuid:
ItemUuid::from_string(attributes.item_uuid),
 862                         });
 863
 864                     op_model::SsoLoginAttributes {
 865                         provider:
op_model::SsoLoginProvider::from_model_string(
 866                             attributes.provider,
 867                         ),
 868                         item,
 869                     }
```

Dynamic tests proved that this impression is correct and arbitrary existing vault and item UUIDs of the same account can be specified. This way, a user importing a malicious 1PUX file could be tricked to select an attacker-controlled login item for the specified Web page.

Further, the import can be used by an attacker to store a respective item to a shared vault of their own account. Since the reference can point to any item in the same account, it can target non-shared vaults of other users.

The likelihood of a successful attack is further increased, since the SSO account selection view of the browser extension does not show information on the source vault (similar to the UI shortage described in section 5.1):

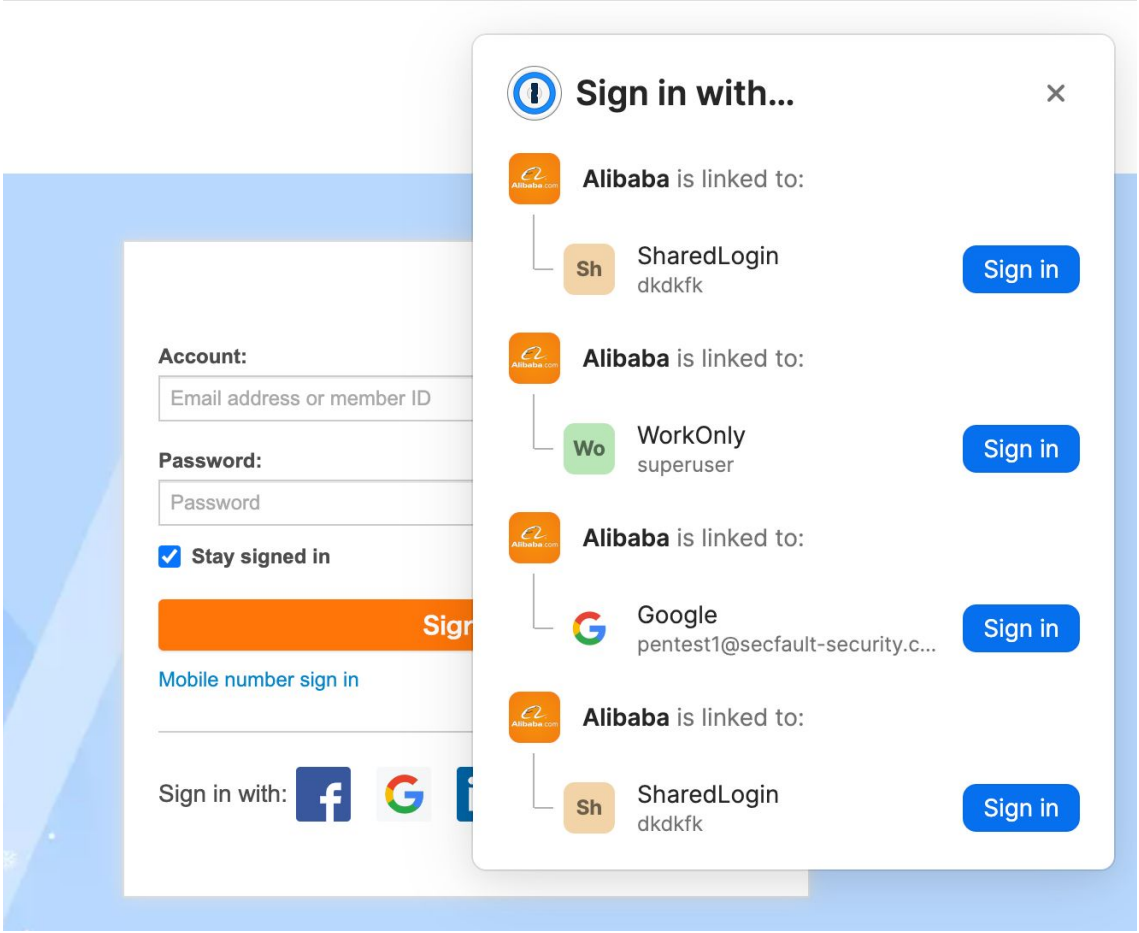

Figure 3 - Selection View for SSO Accounts

Recommended Action

Similar to the solution applied for deserializing linked related items, the vault and item UUID should be mapped to the freshly created vault and item UUIDs created during the import. This way, no references pointing outside the new contents would be possible. In case such a mapping cannot be found, e.g. because it pointed to vault note included in the import due to missing export permissions, the reference should either be removed from the created item completely or the user could manually select a login item that should be used for the Web page instead.

Reproduction Steps

In order to reproduce this issue, please proceed as described below.

1 Use the browser extension to generate a login item with a "single sign-on" field by authenticating to a Web page offering social media account logins, e.g. `https://passport.alibaba.com`. Save both the utilized test SSO account and select it to be used for this page in future.

2 Export the respective 1Password account to a 1PUX file.

3 Extract the 1PUX file as a ZIP archive and open the `export.data` file in a text editor.

4 Repeat step 2 and 3 with the account that should be attacked and must have at least one login item, ideally in a shared exportable vault. Those will be denoted as target account and target login item hereinafter.

5 Perform a textual search in the second `export.data` file for the target login item and the respective vault using their names. Note their UUIDs.

6 Search for the SSO login item in the data of the first export. Now exchange the `vaultUuid` and `itemUuid` JSON field values as shown above in the "Details" section with the noted UUIDs.

7 Zip the contents of the first export after this modification to a file with the extension `.1pux`.

8 Import the created archive to the target account.

9 In the client of the target account search for the freshly created vault and the SSO login item in it. Open its detail view and observe that the target login item is linked although it is in none of the new vaults. The browser extension should now offer the login via the target item to the respective Web page.

4.9 Vault Permission Preview Incorrect

Summary

Type	Location	Severity
Observation	Web UI	N/A

Technical Description

In the context of testing the new 1Password 8 data import feature, it was observed that shared vaults can be excluded from exports for a Business account. As part of the vault permission settings, a "Client Settings" area is displayed. Hereby, the client can be instructed to not include the respective vault in an export file.

When using this setting, it was observed that the short permission and settings preview that is shown before opening the actual management subview is incorrect. On unchecking the "Client Settings -> Export items" box, the "Export" permission is still shown in the preview.

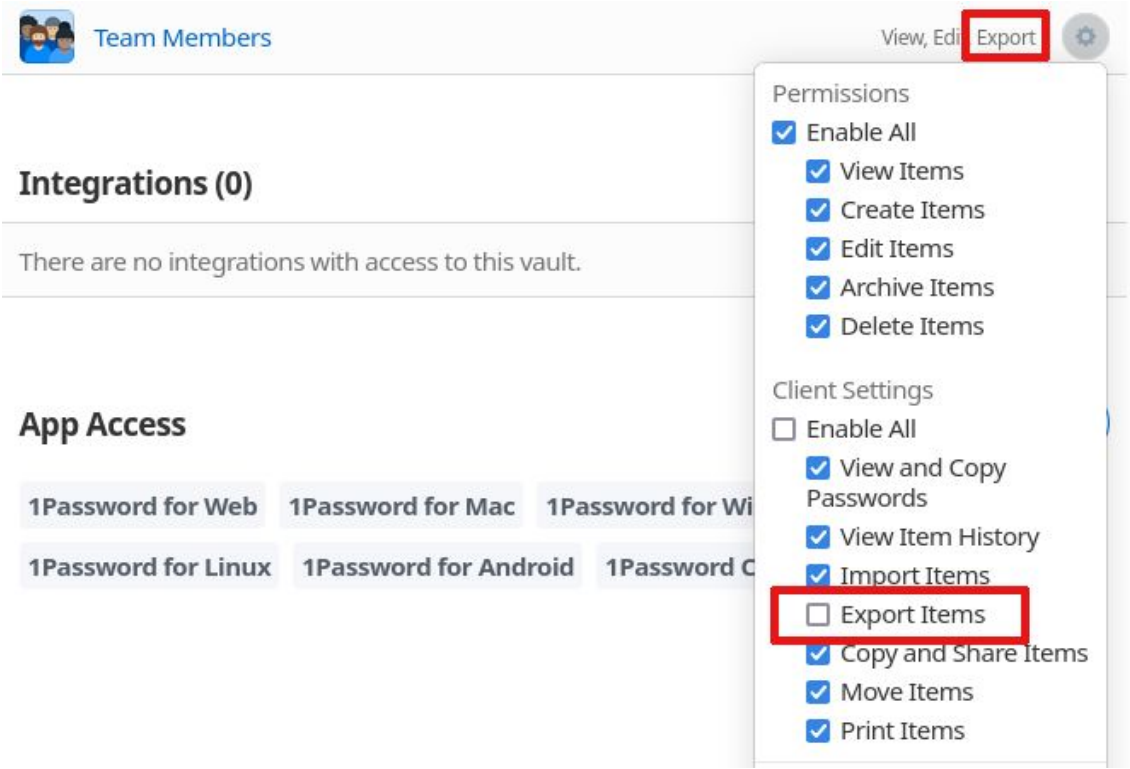

Figure 4 - Incorrect Permission Preview

The issue was found to be present across page reloads and new logins.

No direct security impact is considered to be entailed by the wrong preview of the export permission. Since the Web UI was not part of the assessment's scope, the cause of this flaw was not

investigated. It might therefore be possible that it affects the display of other permissions, which might result in undesired vault security settings.

Recommended Action

The reason for the inaccurate generation of the vault permission preview should be determined, to resolve it in a manner that rules out similar flaws in the context of other access settings.

Reproduction Steps

In order to reproduce this issue, please login to an account privileged to configure vault access permissions for a Business account via the Web UI. Navigate to the vault details page via the right sidebar selecting "Vaults" and afterwards the respective target vault. The URL in the address bar should have the path `/vaults/details/<vault_uuid>`.

Please note the permission preview that is shown for each entry in a "People" or "Group" row. The permissions can be altered by clicking on the settings icon right beside the preview. Here select and remove the "Export item" client setting and observe that it has no impact on the preview. Reload the page or perform a new login to the account to verify that the display does not get updated.

5 Additional Observations

Secfault Security would like to point out a number of general observations and recommendations regarding the analyzed system in the following subsections.

5.1 Vault Display for Favorites

On inspection of the data import feature, it was noticed that imported items are always stored in freshly created vaults. This provides a clean separation between already existing items and imported content, addressing several security aspects at once. While analyzing the solution for possible attack scenarios, such as overwriting existing items, it was however found that the client feature allows for marking items as favorites. This results in such items being listed in a separated overview.

For the MacOS client display of this list, it was observed that neither account nor vault information is shown before a user actively selects the respective item. This way the likelihood of item confusions is increased, which could lead to user mistakes especially in the context of business or team accounts. Please also refer to issue 4.1 that outlines an attack path that could allow to mark imported items a favorites.

5.2 CSV Balanced Quote Check

On importing a CSV file to the MacOS client, it is checked in a first step that the contained quotes are balanced (implemented inside `op-import-export/src/models/csv.rs` in the function `check_balanced_quotes`). It was noted that quotes escaped by a preceding backslash are not considered as quote meta characters. This however omits situations where quotes are prefixed by an even number of backslashes, whereby those escape themselves and must not be considered to escape the quote.

To reproduce the issue import a CSV file with the following content and observe that it is not rejected, although it contains an uneven number of not escaped quotes in a line:

```
Title,Url,Username,Password,OTPAuth,Favorite,Archived,Tags,Notes
\\"LoginJ,,someuser,somepw,,false,false,,
```

No direct security impact is considered to be introduced by this, since the login item was found to be correctly parsed except for the `Title` field. It might however result in rendering legit CSV import files unusable.

5.3 Backslashes in Items break CSV Export Files

According to the balance check performed on quotes contained in imported CSV files, the MacOS client was found to generate broken CSV export files. This is caused by backslashes being copied to the CSV export literally, when contained in an item, while the check considers those as escape

characters. It was observed that the CSV export of a item including the character sequence `\"` will result in an unbalanced quotes error on import.

To retrace the issue create a login item with the title `Title\"` and create a CSV export for the respective account afterwards. Now import this file and observe the displayed error message.

6 Customer Feedback

After receiving a draft version of this document, Agilebits reviewed the identified issues and provided feedback, describing their assessment. In order to provide full transparency, this feedback is included in the below sections.

6.1 Unprotected Item Fields in 1PIF2 Format (Finding 4.1)

1Password accepts this finding as a best practice issue given the impacted format is currently unsupported. We will review this finding for implementation prior to when the format is slated to be supported.

6.2 CSV Injection In Export (Finding 4.2)

1Password accepts this issue and will work to implement a fix in the future. We consider the issue low severity given a malicious actor would only be able to affect a single client at a time and it requires access to a vault the target user would then have to export the data from.

6.3 Questionable Security Advantage of DSecret MFA (Finding 4.3)

With 1Password, MFA is about device trust. MFA is not required upon every sign-in, but only once on every new device that has been set up. As a result, considerations that apply to other MFA implementations don’t necessarily translate to our MFA design. We believe that Secfault security applied considerations that apply for other MFA implementations. However, since we are interested in making MFA with 1Password more powerful than it is today, we have decided to accept this issue as a low severity issue.

6.4 Master Password and Secret Key in Export Files (Finding 4.4)

1Password appreciates the observations made by Secfault within this finding and we do believe users should be made aware these items are included in their export. The finding itself falls into a best practice classification and would be considered as a product enhancement in the future.

6.5 Unsafe chown Call (Finding 4.5)

We accepted this issue and have implemented a fix.

6.6 Insufficiently Authorized MFA Management (Finding 4.6)

With 1Password, MFA is about device trust. MFA is not required upon every sign-in, but only once

on every new device that has been set up. As a result, considerations that apply to other MFA implementations don’t necessarily translate to our MFA design. We believe that Secfault security applied considerations that apply for other MFA implementations. We are generally interested in making MFA with 1Password more powerful than it is today, but within restrictions of current product features we have decided not to accept this finding.

6.7 Unsafe Client Storage of Secret Key (Finding 4.7)

1Password does not accept this finding. The Secret Key is meant to add entropy to 1Password's encryption for data stored on 1Password's servers. Locally on devices, your account password protects access to your vaults. See the 1Password Security Design Whitepaper under the section titled "Locally exposed Secret Keys" for more information.

6.8 Import of References to Existing Vaults (Finding 4.8)

1Password accepts this issue and will address it in the future.

6.9 Vault Permission Preview Incorrect (Finding 4.9)

We agree with Secfault’s observation that this is a bug with no security impact. We’ve started tracking this internally for a bug fix.

7 Vulnerability Rating

This section provides a description of the vulnerability rating scheme used in this document. Each finding is rated by its type and its exploitability/impact of a successful exploitation. The meaning of the individual ratings are provided in the following sub-sections.

7.1 Vulnerability Types

Vulnerabilities are rated by the types described in the following table.

Type	Description
Configuration	The finding is a configuration issue
Design	The finding is the result of a design decision
Code	The finding is caused by a coding mistake
Observation	The finding is an observation, which does not necessarily have a direct impact

7.2 Severity

The severity of a vulnerability describes a combination of the likelihood of attackers exploiting the vulnerability, and the impact of a successful exploitation.

Severity Rating	Description
Not Exploitable	This finding can most likely not be exploited.
Low	The vulnerability is either hard to exploit (e.g., because a successful exploitation requires significant prerequisites) or its consequences can be considered benign.
Medium	The vulnerability can be exploited (possibly under certain preconditions) and a successful exploit can be used to at least partially bypass the security guarantees of the solution.
High	The vulnerability can be exploited easily and a successful exploit bypasses one of the core security properties of the solution.
Critical	The vulnerability can be exploited easily and a successful exploit can be used to compromise systems beyond the scope of the analysis.

8 Glossary

Term	Definition
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
CSV	Comma Separated Value
GCM	Galois Counter Mode
JSON	JavaScript Object Notation
MDM	Mobile Device Management
MFA	Multi-Factor Authentication
OS	Operating System
OWASP	Open Web Application Security Project
SSH	Secure Shell
SSO	Single Sign-On
TOTP	Time-based One-Time Password
URL	Uniform Resource Locator
UUID	Universally Unique Identifier
ZIP	ZIP (compressed archive file format)