Secfault Security

OP Security Assessment

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	2022-02-23	First Draft
0.2	Gregor Kopf	2022-03-01	Additions and Corrections
0.3	Maik Münch	2022-03-03	Internal Review
0.4	Dirk Breiden	2022-03-03	Customer Version
0.5	Gregor Kopf	2022-03-16	Included Customer Feedback
0.6	Gregor Kopf	2022-04-04	Included Additional Customer Feedback
0.7	Gregor Kopf	2022-04-08	Minor Changes in Customer Feedback
1.0	Gregor Kopf	2022-04-27	Final Version

Table of Contents

1 Executive Summary....4
2 Overview....5
2.1 Target Scope....5
2.2 Test Procedures....5
2.3 Project Execution....5
3 Result Overview....7
4 Results....8
4.1 CLI Tools....8
4.1.1 Lax Parsing for Dotenv Files....8
4.1.2 Output of Escape Sequences....10
4.1.3 Race Condition in File Creation....12
4.1.4 Access to Parent Environment in op run....15
4.1.5 Missing Privilege Dropping in op run....17
4.1.6 Secret Data in Command Arguments....18
4.1.7 ToCToU Weakness in Windows Peer Verification....19
4.2 SSH Agent....20
4.2.1 Insufficient Peer Verification on Unix Sockets....20
4.2.2 Unauthenticated Meta Information in Database....23
5 Additional Observations....26
5.1 SSH Key Title Uniqueness....26
5.2 SSH Key Title Manipulations....26
5.3 SSH Private Key Export....27
5.4 Prompt for Unknown SSH Key....28
6 Customer Feedback....30
6.1 Lax Parsing for Dotenv Files (Finding 4.1.1)....30
6.2 Output of Escape Sequences (Finding 4.1.2)....30
6.3 Race Condition in File Creation (Finding 4.1.3)....30
6.4 Access to Parent Environment in op run (Finding 4.1.4)....30
6.5 Missing Privilege Dropping in op run (Finding 4.1.5)....30
6.6 Secret Data in Command Arguments (Finding 4.1.6)....30
6.7 ToCToU Weakness in Windows Peer Verification (Finding 4.1.7)....31
6.8 Insufficient Peer Verification on Unix Sockets (Finding 4.2.1)....31
6.9 Unauthenticated Meta Information in Database (Finding 4.2.2)....31
7 Vulnerability Rating....32
7.1 Vulnerability Types....32
7.2 Exploitability and Impact....32
8 Glossary....34

1 Executive Summary

Secfault Security was tasked by Agilebits with a security review of selected components of the 1Password ecosystem, namely the CLI integration and the SSH agent functionality. The review has been performed in the time frame from 2022-02-14 to 2022-03-01. 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. One of the more severe issues might allow sufficiently-privileged local attackers to trick the user into granting SSH agent authorization for malicious processes. Other issues include problems with the way the CLI handles the execution of external commands, which could result in malicious processes started by the CLI being able to obtain access to the user's 1Password session. Additionally, a number of minor problems have been identified, including for instance the insecure handling of terminal escape sequences or the use of lax parsers.

Section 5 of this document provides a number of additional observations and recommendations. This includes an approach for addressing the known security issue concerning the manipulation of SSH key titles as shown in the approval prompt.

Overall, the reviewed codebase left a positive impression. The code is well-structured and readable and has been implemented with security 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.

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 selected parts of the CLI implementation and the SSH integration of the solution.

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 following source code repositories have been provided by Agilebits for review:

- `op` at revision `7398e6be5c4d359ecb2d52a5116c2b66d7ccad9f`
- `core` at revision `22e7c45a8cffbbaf79be1510cfe24b955c1421a9`

Furthermore, Agilebits provided the respective binaries for the above mentioned revisions.

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 2022-02-14 to 2022-03-01 in 15 person days.

The consultants assigned to this projects were:

- Maik Münch

- Jennifer Gehrke
- Gregor Kopf

3 Result Overview

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

Description	Chapter	Type	Exploitability	Attack Impact
Lax Parsing for Dotenv Files	4.1.1	Code	Medium	Medium
Output of Escape Sequences	4.1.2	Code	Low-Medium	Medium
Race Condition in File Creation	4.1.3	Code	Low-Medium	Medium
Access to Parent Environment in op run	4.1.4	Code	Medium	High
Missing Privilege Dropping in op run	4.1.5	Code	Medium	High
Secret Data in Command Arguments	4.1.6	Code	Low-Medium	Medium
ToCToU Weakness in Windows Peer Verification	4.1.7	Code	Medium	High
Insufficient Peer Verification on Unix Sockets	4.2.1	Design	Medium	High
Unauthenticated Meta Information in Database	4.2.2	Design	Medium	Medium

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 CLI Tools

4.1.1 Lax Parsing for Dotenv Files

Summary

Type	Location	Exploitability	Attack Impact
Code	godotenv (third-party dependency)	Medium	Medium

Technical Description

While reviewing the dotenv file parsing of the `op inject` command, it was found that the used parser implementation is rather lax. A cursory inspection of the parser code[1] revealed that - for instance - the handling of comments does not appear to properly address situations where multiple quotes occur in a nested fashion:

```go
func parseLine(line string, envMap map[string]string) (key string, value 
string, err error) {
	if len(line) == 0 {
		err = errors.New("zero length string")
		return
	}

	// ditch the comments (but keep quoted hashes)
	if strings.Contains(line, "#") {
		segmentsBetweenHashes := strings.Split(line, "#")
		quotesAreOpen := false
		var segmentsToKeep []string
		for _, segment := range segmentsBetweenHashes {
			if strings.Count(segment, "\"") == 1 || strings.Count(segment, 
"'") == 1 {
				if quotesAreOpen {
					quotesAreOpen = false
					segmentsToKeep = append(segmentsToKeep, segment)
				} else {
```

1 https://github.com/joho/godotenv/blob/c40e9c6392b05ba58e6fea50091ce35a1ef020e7/godotenv.go#L100

```
                              quotesAreOpen = true
                        }
                  }

                  if len(segmentsToKeep) == 0 || quotesAreOpen {
                        segmentsToKeep = append(segmentsToKeep, segment)
                  }
            }

            line = strings.Join(segmentsToKeep, "#")
      }
```

It can be observed that the code would treat a string like `"'" # foo` as one actual literal - contrary to what one might expect.

An attacker with the ability to create dotenv files might abuse this behaviour in order to craft seemingly benign dotenv files, which would inadvertently leak secret information into unrelated environment variables.

It should be noted that no in-depth review of the parser implementation has been performed and that the presence of other issues can hence not be ruled out.

Recommended Action

In order to address this issue, a first step could be to perform a more in-depth review of the used dotenv parser in order to identify further possible issues. All identified problems should subsequently be reported upstream in order to be addressed by the developers of the library.

Reproduction Steps

In order to reproduce this issue, please install godotenv and create a dotenv file `/tmp/test.env` with the following contents:

```
PASSWORD=super secret
USERNAME='as908dzf/"' # Has to be quoted, because we hard-code it. When using
variable references like $PASSWORD, this is not needed
```

Then, please use a command as shown below to observe the behaviour of the parser:

```
λ go/bin/godotenv -f /tmp/test.env env
USERNAME='as908dzf/"' # Has to be quoted, because we hard-code it. When using
variable references like super secret, this is not needed
PASSWORD=super secret
```

It can be observed that the `USERNAME` variable - contrary to what one might expect - not only contains the full comment, but also exposes the contents of the `PASSSWORD` variable within this comment.

4.1.2 Output of Escape Sequences

Summary

Type	Location	Exploitability	Attack Impact
Code	CLI Tools	Low-Medium	Medium

Technical Description

While reviewing the implementation of the CLI tools, it was found that the `op` tool generally does not filter terminal escape sequences when writing data to `stdout`. An attacker with the ability to control (parts of) the tool's output could therefore inject malicious escape sequences.

Depending on the used terminal emulator, this can lead to a number of possible issues. Historically, there have been code execution vulnerabilities in a number of terminal emulators, such as xterm or more recently in xterm.js[2].

However, even without such issues, outputting untrusted escape sequences can result in potential problems. Attackers could for instance use escape sequences in order to display misleading information to the user. One obvious example of such misleading information could be fake password prompts, aiming to trick the user into entering sensitive passwords into their command shell.

Recommended Action

In order to address this issue, it is recommended to filter terminal escape sequences when the `op` tool interacts with a tty.

Reproduction Steps

In order to demonstrate the presence of the problem, the following Python script can be used to generate a file named `poc.json`, which contains a sequence of terminal escape sequences aiming to trick a user into entering their password:

```python
import sys

with open('poc.json', 'w') as f:
    f.write(u"\u001b[?25l\u001b[2J\u001b[1;1HPlease enter your password:")
    for i in range(1000000):
        f.write(u"{}\u001b[50D".format(' '*50))
```

The file `poc.json` can subsequently be added to a vault as shown below:

```
$ op document create poc.json
```

2 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0542

After adding the file, the target user could be tricked into displaying the file:

```
$ op document get poc.json
```

The escape sequences in the above PoC code will display a password prompt, which could make the user assume they are still interacting with the `op` tool - while in reality, they would enter their password into their command shell, which in turn might for instance save it in the history file.

4.1.3 Race Condition in File Creation

Summary

Type	Location	Exploitability	Attack Impact
Code	op-cli/command/opio/ file.go	Low-Medium	Medium

Technical Description

The CLI tools offer a number of ways for creating files. For instance, the `op inject` command can be used to parse template files, fill in secret information and create new files. While reviewing the file creation logic in `op-cli/command/opio/file.go`, it was identified that the implementation of `CreateFile` contains a number of possible race conditions. Further, it also does not appear to properly consider symbolic links.

Please consider the below excerpt from the code:

```
func CreateFile(name string, fileMode os.FileMode, io Stdinout, force bool)
(io.WriteCloser, error) {
   var file *os.File
   _, err := os.OpenFile(name, os.O_WRONLY, fileMode)
   if err == nil {
      // Return error when in pipe
      if io.IsOutputPiped() && !force {
         return nil, fmt.Errorf("file %s already exists", name)
      }

      if !force {
         prompt := fmt.Sprintf("File %s already exists, overwrite it? [Y/n] ",
name)
         if ok, _ := input.ProcessYesNoReturnPromptRW(prompt, io.TTYOrStdin(),
os.Stderr); !ok {
            fmt.Println("Aborting.")
            return nil, ErrAborted
         }
      }
      // This is necessary in order to wipe the contents of the previously
existing file.
      file, err = os.Create(name)
      if err != nil {
         return nil, fmt.Errorf("could not overwrite file %s: %s", name, err)
      }

   } else if os.IsNotExist(err) {
      file, err = os.Create(name)
      if err != nil {
         return nil, fmt.Errorf("could not create file %s: %s", name, err)
      }
   } else {
      return nil, fmt.Errorf("could not open file %s: %s", name, err)
   }

   err = file.Chmod(fileMode)
   if err != nil {
      return nil, fmt.Errorf("could not set permissions on file at %s: %s",
name, err)
   }
   return file, nil
```

It can be observed that the code first checks if the target file already exists. If it exists, it prompts the user whether to overwrite the existing file. Subsequently, it clears the file's contents and changes the file permissions. It should be noted that these operations are not performed in an atomic manner. For instance, the target file could be created just after the code checks for its existence. Furthermore, if the target file is a symbolic link, the code could write files to unintended locations. Depending on the respective environment, this might turn into an exploitable condition.

One obvious example of such an exploitable condition leverages the missing check for a symbolic link: Assume that a legitimate user Alice uses the `op inject` command to create a file named `/tmp/foo`. The attacker, Bob, knows about this and creates a symbolic link `/tmp/foo` beforehand. The symbolic link points to Alice's `~/.bashrc` or a similar sensitive configuration file. Alice now runs the `op inject` command. As the target of the symbolic link already exists, the code will prompt her to overwrite the file `/tmp/foo`. She confirms this prompt and thereby accidentally overwrites her `~/.bashrc`.

In order to improve the attack, Bob might attempt to create the symbolic link `/tmp/foo` directly after the code checked for the file's presence. In this case, no prompt would be shown to Alice.

It should be noted that the above example relies on the `fs.protected_symlinkssysctl` variable being set to zero. However, on the one hand the value of this variable is not under the control of the analyzed codebase. On the other hand, more involved attacks, which do not rely on `fs.protected_symlinks` being set to zero, cannot fully be ruled out.

Recommended Action

In order to address this issue, the following approach for creating a file is recommended:

1 Attempt to open the file using the `O_CREAT|O_EXCL|O_NOFOLLOW` flags, while directly passing the desired file permissions as well

2 When the result of the above operation is `EEXIST`, prompt the user to overwrite the file. In case of `ELOOP`, inform the user about the fact that a symbolic link has been detected and abort the

process

3. If the user confirms, `unlink` the file and re-start the process at step 1

Reproduction Steps

This finding has been identified in a static source code review and has not been reproduced dynamically. Hence, no reproduction steps are provided.

4.1.4 Access to Parent Environment in op run

Summary

Type	Location	Exploitability	Attack Impact
Code	op-cli/command/run.go	Medium	High

Technical Description

The `op run` command allows users to start processes, passing secrets stored in 1Password via environment variables. In order to prevent the started process from directly interacting with the running OP8 instance, the code removes a number of sensitive environment variables prior to executing the child process.

However, it should be noted that the started child process could still be able to obtain such environment variables from its parent process by reading `/proc/<ppid>/environ`. This could allow malicious child processes to interact with the running OP8 instance.

Recommended Action

In order to address this issue, the `PR_SET_DUMPABLE` attribute of `prctl` could be used to make the `op` process inaccessible by its child processes.

Reproduction Steps

In order to reproduce this issue, please use the following Python script:

```python
#!/usr/bin/env python3

import os

print("My environment:")
for e in os.environ:
    if 'OP_SESSION' in e: print(e)

stat = open("/proc/self/stat").read()
ppid = stat.split(" ")[3]
print("ppid = " + ppid)

penv = open("/proc/" + ppid + "/environ").read()
entries = penv.split('\0')
print("Parent's environment:")
for e in entries:
    if 'OP_SESSION' in e: print(e)
```

When started with `op run`, the script should first display the environment variables it can directly access. Please note that the `OP_SESSION` variable is not set. However, by reading the environment of its parent process, the script can still access the contents of the `OP_SESSION` variable, as the below excerpt illustrates:

```
λ op run python poc.py
My environment:
GNOME_DESKTOP_SESSION_ID
DESKTOP_SESSION
ppid = 2740520
Parent's environment:
DESKTOP_SESSION=regolith
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
OP_SESSION_secfaulttest1=nveh5yNn4NzHQLbPrVm6vQOQf-ch8o0ZoND_GOGPzk8
```

4.1.5 Missing Privilege Dropping in op run

Summary

Type	Location	Exploitability	Attack Impact
Code	op-cli/command/run.go	Medium	High

Technical Description

While reviewing the implementation of the `op run` command, it was found that the `op` tool does not drop its privileges prior to executing its child process. For enabling the new "Biometric Unlock" feature on Linux, the `op` binary belongs to the group `onepassword-cli` and has the `setgid` flag set. This means that the binary will have its group set to `onepassword-cli` even if it is started by a user who is not a member of this group. This mechanism serves the purpose of being able to identify the binary when it communicates with OP8 via a Unix socket.

The fact that the `op` binary does not drop its privileges however means that processes started by `op run` will also have the group `onepassword-cli`, which could enable them to directly interact with the running OP8 instance via its Unix socket.

Recommended Action

During the execution of the project, the issue was communicated to Agilebits, who stated that the issue has already been addressed by dropping privileges in the `op` process.

Reproduction Steps

In order to demonstrate the presence of this issue, the following command can be used:

```
op run -- id -g --name
```

The output of the `id` command started via `op run` indicates, that the child process indeed belongs to the `onepassword-cli` group.

4.1.6 Secret Data in Command Arguments

Summary

Type	Location	Exploitability	Attack Impact
Code	op-cli/command/ signin.go	Low-Medium	Medium

Technical Description

While reviewing the implementation of the `op` CLI tool, it was found that the `op signin` command accepts possibly sensitive information via command line arguments. The syntax for invoking the command is `signin [<sign_in_address> [<email_address> [<secret_key>]]]`, which indicates that the `secret_key` for a user account could be provided on the CLI. This might leak the secret key info the user's shell history, as well as in the system's process list (e.g., accessible via the `ps` command).

Recommended Action

In order to address this issue, it is recommended to at least inform users about the potential risks of passing data as command arguments. If using command arguments is required for convenience, it could be advisable to process such arguments only if a specific flag (such as `--insecure`) has been provided.

Reproduction Steps

In order to reproduce the issue, please use the `op signin` command, providing the `secret_key` as a parameter. Then, please inspect the process list of the system, as well as the respective user's shell history file.

4.1.7 ToCToU Weakness in Windows Peer Verification

Summary

Type	Location	Exploitability	Attack Impact
Code	CLI Tools	Medium	High

Technical Description

The Windows CLI tools connecting to a named pipe to communicate to the main application are verified based on Authenticode signatures. The signature of the executable is checked and its issuer is ensured to belong to AgileBits. This procedure consists of multiple steps:

1. First, the PID of the connecting process is determined using the API function `GetNamedPipeClientProcessId`
2. The execution path of the process is requested by a call to `QueryFullProcessImageNameW`
3. The file at the execution path is opened
4. The execution path is provided as input to `WinVerifyTrust` to perform the Authenticode verification
5. The subject of the signature issuer, as retrieved from the trust store, is checked to belong to AgileBits

As known to Agilebits this validation steps might be affected by Time-of-check-to-time-of-use (ToCToU) problems. For this reason, step 3 was implemented. It should lock the respective file, to prevent it from being renamed and overwritten before it is read as part of step 4 and 5. This should rule out that an attacker performs a connection with a malicious binary "A" and renames the CLI tools binary to binary "A" (and binary "A" to something else) directly after step 2. Thereby, the legitimate binary would be used in step 4 and 5 although it is not the actual executable.

The implementation of this approach, however, suffers from an implementation issue. It should be noted here, that this perception was gained by a static code review and was not confirmed dynamically.

The issue arises from the selection of the function `std::fs::File::open`[3] for acquiring a file lock. An inspection of its source code[4] showed the internal utilization of `std::fs::OpenOptions::open`:

```
pub fn open<P: AsRef<Path>>(path: P) -> io::Result<File> {
    OpenOptions::new().read(true).open(path.as_ref())
}
```

A further cursory inspection of the documentation for the Windows `std::fs::OpenOptions`

3 https://doc.rust-lang.org/stable/std/fs/struct.File.html#method.open
4 https://doc.rust-lang.org/stable/src/std/fs.rs.html#327-329

extension revealed the default sharing mode[5] requested for the file:

> *By default share_mode is set to FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHARE_DELETE. This allows other processes to read, write, and delete/rename the same file while it is open. Removing any of the flags will prevent other processes from performing the corresponding operation until the file handle is closed.*

As the implementation of `std::fs::File::open` does not explicitly use the function `std::os::windows::fs::OpenOptionsExt::share_mode` to clear all flags (note the example in the documentation[6]), it is expected to utilize the mentioned defaults. As a result, it should still be possible to rename and overwrite the file after step 3.

On a successful attack, the connection of an attacker-controlled binary would be accepted by the named pipe to make requests on behalf of the CLI tools.

Recommended Action

Generally, a more in-depth analysis of the situation is recommended. The missing file locking should be added by using appropriate `open` flags on the Windows platform.

Reproduction Steps

This issue has been identified during a static source code review and has not been reproduced dynamically. Hence, no reproduction steps can be provided.

4.2 SSH Agent

4.2.1 Insufficient Peer Verification on Unix Sockets

Summary

Type	Location	Exploitability	Attack Impact
Design	ssh/op-ssh-agent/src/lib.rs	Medium	High

Technical Description

While reviewing the SSH agent socket communication, it was found that on Unix systems, when receiving a new incoming message to the SSH Agent socket, the peer process is determined based on its PID. Please consider the following excerpt from `ssh/op-ssh-agent/src/lib.rs`:

```
op_log::debug!("connection received");
let callback = shared_callback.clone();
let authorized_sessions = shared_authorized_sessions.clone();
```

5 https://doc.rust-lang.org/stable/std/os/windows/fs/trait.OpenOptionsExt.html#tymethod.share_mode
6 https://doc.rust-lang.org/stable/std/os/windows/fs/trait.OpenOptionsExt.html#tymethod.share_mode

```
let handler = tokio::time::timeout(Duration::from_secs(30), async move {
    while let Some(msg_event) = incoming.next().await {
        let message = match msg_event {
            Ok(msg) => msg,
            Err(e) => {
                // If we can't get a message, then we can't do anything else
for this
                // specific connection.
                op_log::warn!(
                    "failed to receive agent request({}), dropping client",
                    e
                );
                break;
            }
        };

        let sock_pid = incoming.pid();
```

The pid function is implemented inside `op-ipc/src/ipc/unix.rs` and utilizes the `peer_cred` function offered by the `tokio::net::UnixStream` struct:

```
pub fn pid(&self) -> Option<u32> {
    let peer_creds = self.0.get_ref().peer_cred().log_err().ok()?;
    let pid = peer_creds.pid()?;
    pid.try_into().ok()
}
```

The documentation states that this "Returns effective credentials of the process which called connect or socketpair"[7]. However, sockets can be shared with other processes by sending the respective file descriptor via a Unix socket. As already hinted by the documentation, a socket message must therefore not originate from the process that initially established the connection. This can lead to a wrong PID value and thereby to wrong process information lookups during the approval process.

As part of the assessment it was analysed whether this behavior can be misused by an attacker to reuse the key approval given for another process. The core problem in the reviewed implementation is the assumption that the process establishing the socket connection still needs to be running. This is however not necessarily the case, as a transferred socket remains usable even after termination of the process that originally opened the file descriptor. As a result, it is possible to store the socket for a certain time in a second attacker-controlled process and wait for an approved process to run with the same PID as the process that originally opened the connection. Since any generation of the `op_sys_info::ProcessInformation` for this PID will now utilize the information for the legitimate process, the socket connection will automatically be approved by the agent's session when sending signature requests.

7 https://docs.rs/tokio/0.1.12/tokio/net/struct.UnixStream.html#method.peer_cred

The approach was dynamically tested on a system with a reduced maximal number of PIDs and ensuring the rapid reuse of the relevant PID, proving its general applicability. An attacker with access to the SSH agent socket can thereby access keys that get approved for the attacked process.

Recommended Action

In order to address this issue, it is recommended to record the start time of the remote process, when a socket connection is originally established. Once a message is sent to the socket, the start time of the remote process should be checked again; if it does not match the recorded start time, the connection should be aborted.

Reproduction Steps

The generated Proof-of-Concept consists of the following steps:

1. Reduce the maximal number of PIDs for the system
2. Start a main process listening on a Unix socket and storing incoming file descriptors
3. Consume the majority of free PIDs with wait processes
4. For the remaining free PIDs: Open processes connecting to the SSH agent socket and sending the respective file descriptor to the main process.
5. Terminate all processes used in step 4 once the system ran out of free PIDs
6. Start a legit process performing a SSH connection with a key stored in OP (The process should automatically get assigned one of the PIDs already used in step 4)
7. Approve the access to the SSH key requested by the process of step 6
8. Let the main process instantly use the stored file descriptor for the PID of the process of step 6 and request a signature using the key from step 7

The PoC implementation has been delivered to Agilebits separate from this report.

4.2.2 Unauthenticated Meta Information in Database

Summary

Type	Location	Exploitability	Attack Impact
Design	OP Core	Medium	Medium

Technical Description

While analyzing the mapping between the data shown in the approval prompts for the SSH agent or CLI biometric unlock prompts and the actual secrets that become accessible, it was identified that the OP database contains various kinds of unauthenticated metadata. This could allow an attacker with access to the respective file system location to modify this data. While this problem is known for the SSH agent key titles (please refer to section 5.2 for a potential solution to this), it seems to be a more general issue.

Dynamic tests showed, for example, that the account name displayed throughout the client UI and in the CLI biometric prompts can be changed as well as deleted items can be reactivated. Thereby, users can be tricked to approve CLI access to items of unintended accounts and deleted credentials get accessible without any user interaction.

Cursory tests hint that data that is not cryptographically protected locally, e.g. entries consisting of JSON data blobs, can be modified and will become available latest during unlock. This concerns the various plain text data entries, including row references defining the object's general context, but might also be of relevance for the meta data attached to encrypted blobs. Whether any security issues exist in the latter case was not analyzed, as it highly depends on the utilized cryptographic implementations which are out-of-scope for the performed assessment.

The impact was rated to be medium, but only relates to the attack scenario of changing the account display name and to the reactivation of items. The impact of this general issue can be considerably higher depending on the type of data that can be altered.

Recommended Action

The ability to tamper data stored in the OP database should be assessed in more detail. It is advisable to introduce a general routine for ensuring data authenticity or integrity. While it is understood that some data needs to be available even before the user password is provided, the majority of data is loaded on a successful unlock and should therefore be protected by default. Unsecured data should be handled separately and may only be used for this specific use case and when authentication routines or integrity checks cannot be applied.

One example for this is the account display name. It is already provided to the user before the first unlock, but is also used as part of the GUI afterwards. In the second case, it is not necessary to read its value without a previous integrity check. Further, similar to the mitigation proposed against manipulations to the SSH key titles (see section 5.2, the account name can be checked on successful biometric unlock against an authenticated value and be rejected in case of modifications.

It is further considered likely that only little exceptions exist that, similar to the account name for the unlock screen, cannot be covered. Although Secfault Security has no full understanding of the general cryptographic design, the current impression is that after the initial unlock of the OP application, the user password or derived values remain accessible even when the account is locked. In this case, it would be possible to utilize them to check the integrity of data even during this. Before the initial unlock, secondary features such as SSH are not available anyway and do therefore not require access user specific data.

In summary:

- Data should be authenticated by default.
- Only minimal exceptions should be defined explicitly.
- The integrity checks may only be omitted for this data and in cases where the cryptographic material for performing the integrity checks is not available.
- Mitigations should be implemented to detect modifications as soon as possible, optimally before disclosing sensitive data or elevating privileges (see the approach described in section 5.2).

Reproduction Steps

To reproduce this issue the following general procedure could be used:

First, please lock the OP application and run the following command to open the database:

```
sqlite3 ~/.config/1Password/1password.sqlite
```

Now, perform the according changes to the database and unlock the OP application to ensure that the data gets reloaded.

Reactivation of Items
As a preparation, please note the UUID of an item using the feature "Copy item UUID". Now, please follow the general procedure and execute the following SQL command for the noted UUID:

```
UPDATE item_overviews SET trashed=1 WHERE uuid="<noted_UUID>";
```

Finally, it can be observed that the according key is no longer active.

Change Account Display Name
As a preparation, please add two accounts with different names, `<Account_A>` and `<Account_B>`, to the OP application. Then, please enable biometric unlock and sign in to `<Account_A>` with the CLI tools. Now, please follow the general procedure and list the account database entries with the command:

```
SELECT * FROM accounts;
```

Please, note the id in the first column and copy the JSON payload in the third column of the entry for `<Account_A>`. Then, please change the `team_name` JSON field value to that of the entry for `<Account_B>` and afterwards encode the new payload using the following Python script:

```
import binascii

binascii.hexlify(b'<the_payload>')
```

Afterwards, please update the database entry using:

```
UPDATE accounts SET data=x'<hex_payload>' WHERE id=<noted_id>;
```

After unlocking, please run the CLI command `op item list` and observe that a system authentication prompt with the account name of `<Account_B>` is shown. On approval, items of `<Account_A>` will become accessible.

Change Account Sign In URL

Considering the above reproduction steps for changing the account's display name, one can instead modify the account's `sign_in_url` JSON field. Here, please set the URL `http://127.0.0.1`. Now listen to port `80` on localhost, e.g. using `netcat`:

```
root:~$ nc -l 80
```

Now, please proceed with unlocking and running the CLI item query. It can be noted that an incoming connection is shown on port 80. Various other errors will be shown during the use of the account due to later connection attempts to the modified sign in URL.

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 SSH Key Title Uniqueness

In the context of the OP SSH agent, the title is the only information available to a user to identify requests to a specific key during the approval process. Accordingly, the displayed title is critical for the selection of the correct key. During the generation of key material inside the main application, it was noted that keys with the same title can be created. This is possible even inside the same vault and is considered likely to happen, as part of the intended use of the application, when keys are stored in multiple vaults and even accounts.

Secfault Security would like to recommend prohibiting the creation of keys with the same title inside the same vault. In the approval prompt, the key title should be prefixed by the vault name, in case more than one local vault exists.

5.2 SSH Key Title Manipulations

Considering the relevance of key titles as outlined in the last subsection, another change to the current implementation was found to be advisable.

The documentation provided by Agilebits states that it is a known security issue that titles are stored in the file system without additional integrity protection. As a result, an attacker with write access to the specific location can modify the titles that will be displayed in the approval prompt, tricking the user to inadvertently allow access to specific keys.

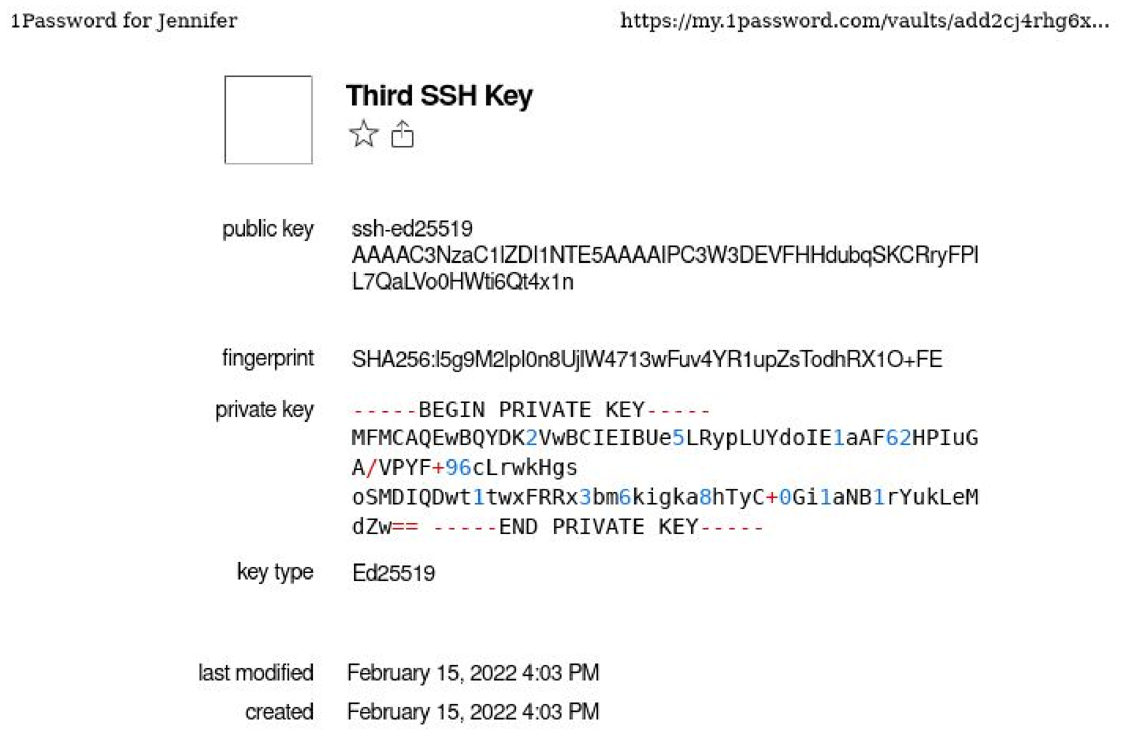
Secfault Security would like to propose the following approach, that requires no additional cryptographic mechanisms: The SSH agent determines the target key by the public key information provided as part of the `SSH_AGENTC_SIGN_REQUEST` request by the SSH client. Now, a lookup in the unencrypted parts of the database is performed to identify the key title, as the vaults might be locked at the moment of the request. This title is then shown in the authorization prompt. On the approval of the user, the vaults need to be unlocked to give access to the respective private key. Therefore, this time it is ensured that a protected version of the key title becomes available. Consequently, it is possible to check the protected key title against the title previously shown in the approval prompt before signing the requested data with the SSH private key. This way the attack is mitigated and modifications to the plain text key titles, and thus attacks on the user, can be detected.

Due to the simplicity of the described approach, it is not recommended to accept the risk of arbitrary manipulations to the plain text key titles.

5.3 SSH Private Key Export

Unlike other common item types, such as passwords, that are supported by the 1Password solution, the use cases for an SSH key pair are considered limited. The public key part of the key pair is not a secret, not requiring special protection with regard to its disclosure, is occasionally configured for long term use on some systems. The private key, on the other hand, is either not exported from the application at all, if it is only used via the offered SSH agent, or is also configured, even more infrequently than the public key, permanently for some clients.

On that account, it was found questionable that the private key can be exported in various ways from the OP client and Web UI. Its plain text can be copied via multiple GUI elements and in different formats to the OS clipboard and is stored in the file system. It is even included in the Web UI's print view:



1Password for Jennifer https://my.1password.com/vaults/add2cj4rhg6x...

Third SSH Key

public key	ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPC3W3DEVFHHdubqSKCRryFPl L7QaLVo0HWti6Qt4x1n
fingerprint	SHA256:l5g9M2lpl0n8UjlW4713wFuv4YR1upZsTodhRX1O+FE
private key	-----BEGIN PRIVATE KEY----- MFMCAQEwBQYDK2VwBCIEIBUe5LRypLUYdoIE1aAF62HPIuG A/VPYF+96cLrwkHgs oSMDIQDwt1twxFRRx3bm6kigka8hTyC+0Gi1aNB1rYukLeM dZw== -----END PRIVATE KEY-----
key type	Ed25519
last modified	February 15, 2022 4:03 PM
created	February 15, 2022 4:03 PM

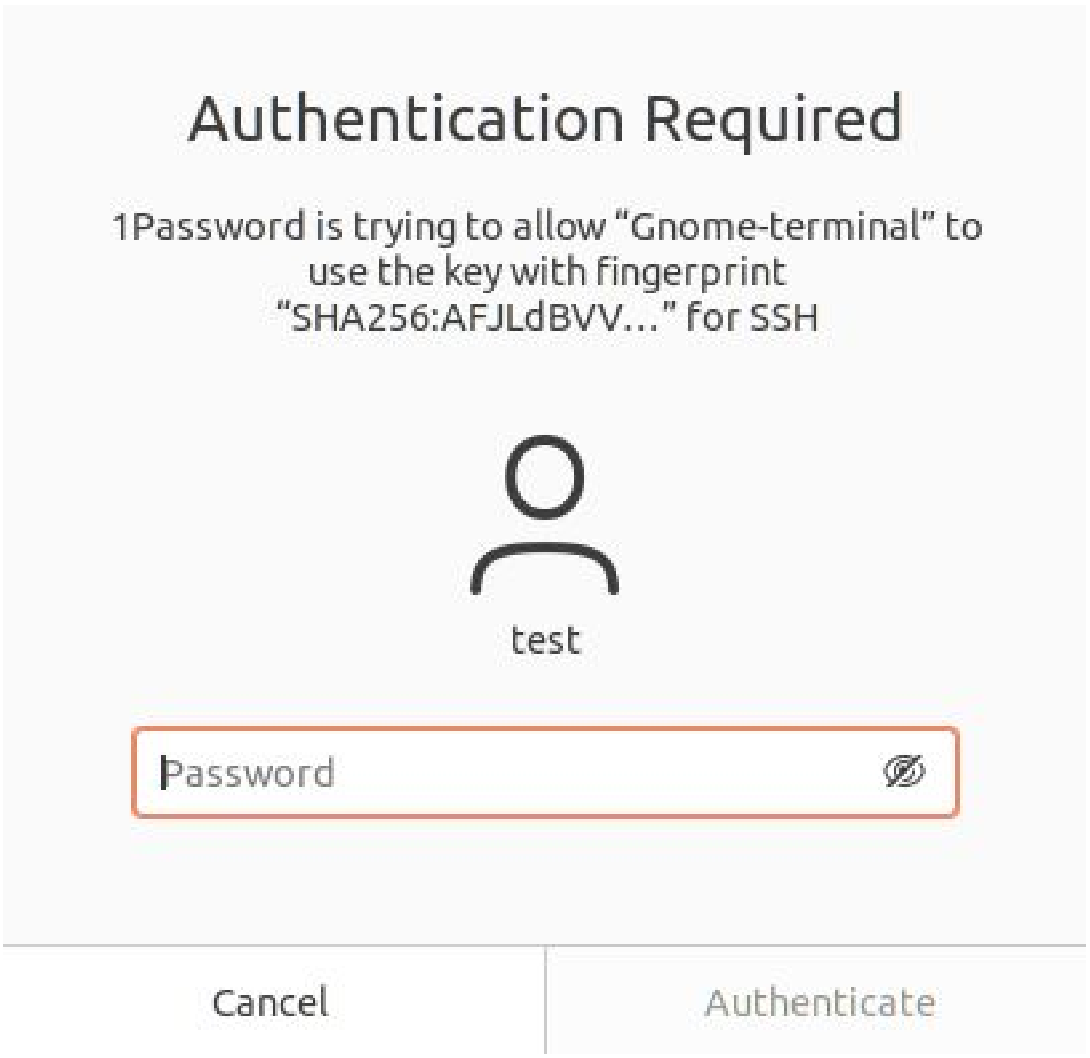
Figure 1 - SSH Key Print View

The described application behavior has two security shortcomings: First of all, in consideration of the normally sporadic export of SSH private keys, the risk of disclosure due to accidental clicks on respective UI elements is considered too high. This is especially the case, as SSH key pairs are intended to be reused for multiple purpose, which is uncommon for other credential types. Further, it is not regarded security best practice to export SSH keys in plain text. Instead, keys should be exported in a wrapped format by default, e.g using PKCS#8 with key encryption based on most recent cryptographic algorithms.

The introduction of a default key wrapping routine involves the implementation of a user prompt that allows to select or generate some passphrase. The same prompt could be used to allow a user to export the key in plain text at their own risk. This change would therefore address both shortcomings, as keys are unlikely to be disclosed accidentally in a multi step process.

5.4 Prompt for Unknown SSH Key

During testing it was observed that the SSH agent will show an approval prompt even if the key specified in the `SSH_AGENTC_SIGN_REQUEST` request is not known. As there is obviously no key title present for it, the SHA256 fingerprint will be shown in the prompt instead:



Authentication Required

1Password is trying to allow "Gnome-terminal" to use the key with fingerprint "SHA256:AFJLdBVV..." for SSH

Figure 2 - Prompt for Unknown SSH Public Key

The following message gets logged on approving the key access with successful system authentication:

```
INFO  2022-02-23T10:37:32.093 tokio-runtime-worker(ThreadId(1)) [1P:ssh/op-ssh-
agent/src/lib.rs:299] Session was not authorized
```

The described behavior could be used by an attacker to confuse the user, which might increase the chance of tricking them to perform more serious actions.

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 Lax Parsing for Dotenv Files (Finding 4.1.1)

We've accepted this finding as a best practice issue. Version 2.0.1 of the 1Password CLI uses a more robust method to parse the environment.

6.2 Output of Escape Sequences (Finding 4.1.2)

We've accepted this finding as a low severity issue. When used in a TTY, version 2.0.1 of the 1Password CLI by default no longer outputs terminal escape sequences such that they are interpreted by the outputting terminal.

6.3 Race Condition in File Creation (Finding 4.1.3)

We've accepted this finding as a low severity issue. Version 2.0.1 of the 1Password CLI has improved file creation logic to avoid this bug.

6.4 Access to Parent Environment in op run (Finding 4.1.4)

We've investigated this finding and noted that our attempt to filter out CLI specific environment variables was ineffective and set the wrong expectations. As a result, version 2.0.1 of the 1Password CLI no longer attempts to filter out the parent environment.

6.5 Missing Privilege Dropping in op run (Finding 4.1.5)

Secfault Security notified us during the test of this issue, and we addressed it promptly. It was addressed in the release of version 2.0.0 of the 1Password CLI, the first release of the op run command.

6.6 Secret Data in Command Arguments (Finding 4.1.6)

We've accepted this finding as a best practice issue. Version 2.0.1 of the 1Password CLI requires explicitly setting a flag to provide the secret key on the CLI directly, in addition to the default method that prompts for it.

6.7 ToCToU Weakness in Windows Peer Verification (Finding 4.1.7)

Secfault Security notified us during the test of this issue, and we addressed it promptly. It was addressed in the release of 1Password for Windows 8.6.0, the first release of the SSH agent.

6.8 Insufficient Peer Verification on Unix Sockets (Finding 4.2.1)

Secfault Security notified us during the test of this issue, and we addressed it promptly. It was addressed in the release of 1Password for Linux 8.6.0, the first release of the SSH agent.

6.9 Unauthenticated Meta Information in Database (Finding 4.2.2)

We've reviewed this issue and can’t accept most of the described findings as valid issues under the threat considerations described in the 1Password Security Whitepaper. However, we are looking to improve some hardening of 1Password configuration data in the future.

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 Exploitability and Impact

The exploitability of a vulnerability describes the required skill level of an attacker as well as the required resources. Therefore, it provides an indication of the likelihood of exploitation.

Exploitability Rating	Description
Not Exploitable	This finding can most likely not be exploited.
Minimal	Although an attack is theoretically possible, it is extremely unlikely that an attacker will exploit the identified vulnerability.
Low	Exploiting the vulnerability requires the skill-level of an expert. An attack is possible, but difficult pre-conditions (e.g., prior identification and exploitation of other vulnerabilities) exist or the attack requires resources not available to the general public (e.g., expensive equipment). Successful exploitation indicates a dedicated, targeted attack.
Medium	The vulnerability can be exploited under certain pre-conditions (e.g., user interaction or prior authentication). Non-targeted, random attacks are possible for attackers with a medium skill level who perform such attacks on a regular basis.
High	The vulnerability can be exploited immediately without special pre-conditions, by random attackers or in an automated fashion. Only general knowledge about vulnerability exploitation is required.

The following table describes the impact rating used in this document.

Impact Rating	Description
Critical	The vulnerability is a systematic error or it permits compromising the system completely and beyond the scope of the assessment.

Impact Rating	Description
High	The vulnerability permits compromising the systems within the scope completely.
Medium	The vulnerability exceeds certain security rules, but does not lead to a full compromise (e.g., Denial of Service attacks)
Low	The vulnerability has no direct security consequences but provides information which can be used for subsequent attacks.
Informational	The observed finding does not have any direct security consequence; however, addressing the finding can lead to an increase in security or quality of the system in scope.

When rating the impact of a vulnerability, the rating is always performed based on the scope of the analysis. For example, a vulnerability with high impact typically allows an attacker to fully compromise one or all of the core security guarantees of the components in scope. Identical vulnerabilities can therefore be rated differently in different projects.

8 Glossary

Term	Definition
API	Application Programming Interface
CLI	Command Line Interface
GUI	Graphical User Interface
JSON	JavaScript Object Notation
OS	Operating System
PoC	Proof-of-Concept
SQL	Structured Query Language
SSH	Secure Shell
URL	Uniform Resource Locator
UUID	Universally Unique Identifier