Pentest-Report 1Password B5 & Permissions 10.2020

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, MSc. R. Peraglie, Dr. N. Kobeissi

Index

Introduction

> *"The easiest and safest way to share logins, passwords, credit cards and more, with the people that matter most. Go ahead, forget your passwords – 1Password remembers them all for you."*
>
> From https://1password.com/

This report presents the results of a security assessment targeting the 1Password B5 feature compound, which essentially describes the 1Password web application. Carried out by Cure53 in October 2020, the project encompassed a penetration test and a source code audit. Ten discoveries, including one marked as *High* severity*,* have been made on the 1Password's scope during this assignment.

To give some context, the project was requested by 1Password and executed by four members of the Cure53 team in Calendar Weeks 43 and 44 of 2020. The 1Password B5 web application was tested by Cure53 before, indicating that the project has been completed in the frames of broader cooperation centered on security aspects. Since *1PW-01,* which took place in early 2019, most of subsequent tests focused on the client software or specifically chosen features. In this test, the focus was solidly placed on the B5 web application itself and, most importantly, on the 1Password permission enforcement utilized by the B5 web application in both frontend and backend.

Making sure that the work can be properly structured and cover all relevant aspects of the scope, Cure53 worked with two work packages (WPs) with the following contents:

- **WP1**: Penetration-Tests & Audits against 1Password B5 Web Application UI;
- **WP2**: Penetration-Tests & Audits against 1Password B5 Web Backend & API.

The scope was well-prepared by 1Password and the Cure53 team can make good progress during fourteen person-days dedicated to this project. Good coverage levels and proper research depth were achieved. As usual for engagements carried out by Cure53 for 1Password, a dedicated, private Slack channel was used to connect all involved staff in real-time exchanges. Communications were productive and helpful with all Cure53's questions answered in a prompt and comprehensive manner. Status details and information about the spotted findings, as well as the resulting possible attack scenarios, were shared once confirmed.

Moving on to results, the Cure53 team managed to spot a total of ten findings, as mentioned above. More specifically, five problems were classified to be security vulnerabilities of varying severity levels. The remaining five items can be seen as general weaknesses, typically - but not entirely in this case- characterized by lower exploitation potential. Of note is the fact that one item overall received marks linked to elevated seriousness. The presence of severe problems among *Miscellaneous* issues can be explained by them being reported before in *PW-01.* Since they have not been fully addressed, Crue53 lists them for completeness' sake. Despite the number and scope of findings, the results are leaning to a positive verdict. This is driven by the fact that especially the focal areas, meaning the enforcement of the 1Password permissions. stood strong against the approach attempted by Cure53.

In the following sections, the report will first shed light on the scope and key test parameters. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this autumn 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations for 1Password B5 web application and 1Password permission enforcement are also incorporated into the final section.

***Note**: This report was updated in early December and notes were added to several tickets to reflect the state of updated severities after the tickets have been reported to and discussed with the 1Password team.*

Scope

- **Penetration-Tests & Source Code Audits against 1Password B5 Web Application**
  - **WP1**: Penetration-Tests & Audits against 1Password B5 Web Application UI
    - Tests performed locally on https://my.b5local.com:3000
    - The application must, therefore, be built on the test-machine from the supplied sources to be able to run it locally.
    - More information can be found in the provided *instructions.md* from the B5 sources
  - **WP2**: Penetration-Tests & Audits against 1Password B5 Web Backend & API
    - Tests performed locally on https://my.b5local.com:3000
    - Note that this engagement aimed at a holistic and thorough review of the 1Password permission enforcement, covering the review of cryptography-, server-, and client-enforced policies
  - **Sources were shared with Cure53**
  - **Test-supporting material was shared with Cure53**

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *1PW-06-001*) for the purpose of facilitating any future follow-up correspondence.

1PW-06-003 API: Blocking users from registration through rate-limiting *(Low)*

***Note**: Upon review 1Password disagreed with Cure53's description of the working of the rate limiting. As a result, 1Password believes the presented risk for brute force attacks is significantly lower than originally presented and requested the severity to be set to low. 1Password will take Cure53's assessment into account to further improve its rate limiting and blocking infrastructure in the future.*

During the assessment of the B5 web application, it was found that the implemented protection against brute-force attacks on the *sign-up* verification endpoint could be used to prevent users from registering with the application. If more than 25 requests are sent to the endpoint within a short amount of time, the server responds with the status *429 Too many Requests* (see below).

At first glance, it appears to be a normal rate-limiting but, in fact, the limit is not bound to the IP of the client but rather to the email address sent in the request. This makes it possible for an attacker to prevent other users from finishing the verification step while sending more than 25 requests with one email address to the affected endpoint.

After the unlock time has passed, the attacker can still send new requests to permanently prevent users from registering with the application. Please note that this issue is reproducible on both the test- and the production environments.

Affected File:
b5/server/src/logic/action/signup.go

Affected Code:

```
func GetSignupDetails(acs *access.UnverifiedAccess, param string, email string,
accountType string, domain string) (*api.Signup, api.StatusCode) {
[...]
   recentSignups, err := acs.SelectRecentSignupsForEmail(email)
   [...]
   for _, recentSignup := range recentSignups {
      if recentSignup.Attempts >= constraints.MaxCurrentSignupTokenAttempts {
         [...]
         acs.LogError("GetSignupDetails attempt blocked due to too many signup
attempts", util.ObfuscateEmail(email))
         return nil, api.StatusTooManyRequests
```

PoC Request:

```
GET /api/v2/signup/000024?email=seba%2B1pwtest5%40cure53.de&account-type=I HTTP/
1.1
Host: my.b5local.com:3000
[...]
```

Response (after 25 requests):

```
HTTP/1.1 429 Too Many Requests
[...]

{"errorCode":121,"errorMessage":"Too many requests","requestId":277195}
```

Steps to reproduce on production environment:

1. Open https://start.1password.com/sign-up and enter the email address to block.
2. Then, intercept the verification request (see PoC above) and repeat it several times with randomly created *6-digit* codes in a short amount of time until status code *429* is returned.
3. With a new IP address, open https://start.1password.com/sign-up again and enter the related email to get a verification code.
4. In the final step, enter and send the newly received verification code. The API will respond with status code *429: Too Many Requests*.

It is recommended to stop binding rate-limiting to the email received via the affected request. Instead, it is advised to bind it to the client's IP address. Additionally, a captcha can be considered to diminish the efficiency of this attack.

1PW-06-004 API: Lack of ACL on endpoints returning user-data *(Low)*

During the assessment of the B5 web application, it was found that several endpoints that return user-data are missing additional access control checks. The first affected endpoint on */api/v2/users* permits querying user-data for different contexts via additional *GET* parameters, such as *vault* or *group*. For example, it is possible to query members from a vault that the authorized user does not belong to (see *PoC #1* below). In addition, the endpoint facilitates obtaining all members from a group via the *GET* parameter *group* (see *PoC #2*).

In combination with the third affected endpoint on */api/v2/account* (see *PoC #3* below), which allows all users and groups of a company or team account to be queried, an attacker can create parts from the "*People*" function, even though they are only available to administrators. This enables an attacker to get sensitive information - such as email addresses - from user-accounts if 2FA is enabled. This also concerns account-types and related UUIDs. The data can be used by an attacker for further exploitation, such as for Phishing campaigns or in combination with other vulnerabilities. The following PoC shows how users from a vault can be requested with the user *seba+4pw@cure53.de* who does not belong to the vault.

PoC Request #1 for getting vault members:
```
GET /api/v2/users?vault=2gpmbr7ch7a4xv4ual3y5u37pu HTTP/1.1
Host: my.b5local.com:3000
X-AgileBits-MAC: v1|17|YiaQSlScyEoqALex
X-AgileBits-Session-ID: FOV5OIOE2RHGPD5KDSVLMUBZCM
[...]
```

Response:
```
HTTP/1.1 200 OK
[...]
```

Decrypted data:
```
users: Array(2)
0:
avatar: ""
email: "seba+1pw6@cure53.de"
firstName: "seba6 (admin)"
lastName: ""
name: "seba6 (admin)"
state: "A"
type: "R"
uuid: "WZBPR7QGWFAGVHGI72XOULQ3IY"
1:
avatar: ""
email: "seba+1pw7@cure53.de"
firstName: "seba7"
lastName: ""
name: "seba7"
state: "A"
type: "R"
uuid: "WGT67JA3IFASNCMTE5NKK7QGXY"
```

PoC Request #2 for getting group members from administrator group:
```
GET /api/v2/users?group=vrgt4leuc7nvcijewwn5vfqvwu&attrs=combined-permissions
HTTP/1.1
Host: my.b5local.com:3000
X-AgileBits-MAC: v1|41|ILn6rfYkSuoN3UZ3
X-AgileBits-Session-ID: O3Q3P5EJF5HK5OETLNG3XZF2TE
```

```
[...]
```

Decrypted Response:

```
users: Array(1)
0:
avatar: ""
combinedPermissions: 6665789243175
email: "seba+1pw6@cure53.de"
firstName: "seba6 (admin)"
lastName: ""
name: "seba6 (admin)"
state: "A"
type: "R"
uuid: "WZBPR7QGWFAGVHGI72XOULQ3IY”
```

PoC Request #3 for getting all users and groups from a company account:

```
GET /api/v2/account?attrs=users,groups HTTP/1.1
Host: my.b5local.com:3000
[...]
```

Decrypted Response:

```
{uuid: "ESJTLEZDVJHEXNQLETI75LKNHM", name: "cure53test", type: "B", state: "A",
avatar: ""}
domain: "cure53test-team"
[...]
groups: Array(7)
[...]
2:
activeKeysetUuid: "i5h4zy3cxthaoloyw4kejapxwq"
attrVersion: 1
createdAt: "2020-10-21T08:16:05Z"
desc: "Administration of users, groups, and vaults."
name: "Administrators"
permissions: 268435204
state: "A"
type: "A"
updatedAt: "2020-10-21T12:11:59Z"
uuid: "vrgt4leuc7nvcijewwn5vfqvwu"
[...]
users: Array(3)
[...]
0:
activities: null
attrVersion: 3
combinedPermissions: 6665789243191
createdAt: "2020-10-21T08:16:04Z"
devices: null
email: "seba+1pw6@cure53.de"
```

```
firstName: "seba6 (admin)"
hasMFAEnabled: false
keysetVersion: 7
keysets: null
language: "en"
lastAuthAt: "2020-10-23T12:58:09Z"
lastName: ""
memberships: null
name: "seba6 (admin)"
newsletterPrefs: 0
personalItemsCount: null
preferences: 16
state: "A"
type: "R"
updatedAt: "2020-10-21T12:22:22Z"
uuid: "WZBPR7QGWFAGVHGI72XOULQ3IY"
vaultAccess: null
[...]
```

It is recommended to make sure that contents can be obtained only by the properly authorized users. This means that implementing further checks is unavoidable. Revised approaches should examine if a user with the requested session is allowed to get the contents of an entry via the specified requests. If this is not the case, the backend should return a corresponding error code.

1PW-06-007 API: Joining members can add arbitrary vaults to account (*High*)

***Note**: Upon review 1Password determined that this was a high priority issue. Despite the attack requiring high pre-existing privileges on behalf of the attacker as well as minor corporation by the victim, even complicated social engineering opportunities must be closed with priority. 1Password resolved the issue immediately.*

It was found that non-guest users that are joining an account via *PUT /api/v1/person/join* could supply a vault, including offering the vault access. The restrictions of the vault and the associated accesses are more relaxed than those applied in the *CreateVault* action that can be triggered by the *POST /api/v{1,2}/vault* endpoints. When joining an account, a personal vault could be added to the account that grants multiple access routes to that vault.

This increases the risk of attackers adding a personal vault to a guest-user's dashboard that is marked as private. At the same time, malicious team members can have access to that personal vault. Guest-users trust 1Passwords hint in that only they have access to their private vault, thus deciding to store sensitive information within that vault. As a result, attackers could steal this sensitive information in plain-text, hence the *High* rating of this finding. Note that the exploit does not require administrators to confirm the attacker but necessitates userIDs in advance.

Steps To reproduce:

1. Invite a *guest* user and a *team* user by email.
2. Sign up and confirm the *guest* user until enrolled.
3. Adjust the exploit script *https://cure53.de/exchange/57958324957752/1PW-06-007.js*:
    1. The *TARGET_UUID* variable should match the targeted user ID.
4. Click on the team user's *invite* link as the attacker and run the exploit script in the window (Paste into F12->Developers Console), tested on *Firefox 81 - 1PW v899.*
5. Proceed to sign up in the same window.
6. The personal vault should be created with *both* parties having vault access.

It is recommended that the *JoinAccount* action that can be triggered by the *PUT /api/v1/person/join* endpoint is improved and calls for strict restrictions on the supplied vault. The restrictions should be at least as tight as applied by the *CreateVault* action. Further, it is advisable that the *JoinAccount* action only allows creation of personal/private vaults.

Generally, when adding a vault through any action, exactly one immutable vault access should be allowed on a personal/private vault. This should grant only the requesting user-access to the vault. By doing so, the attackers cannot create a private vault with multiple access routes for other people and are limited by constant restrictions that cannot be bypassed when joining an account.

1PW-06-008 Crypto: Insufficient public key validation criteria *(Low)*

***Note**: Upon review 1Password noted that its security design does not permit tampering with these keys to mount subsequent attacks. However, 1Password considers proper validation of security parameters important hygiene and has resolved this issue.*

The 1Password server codebase performs a series of public key validation checks for both RSA and EC-based public key-types used by 1Password clients for operations such as vault sharing. However, it was observed that the extent of public key validation was limited to checking whether keys had *"key IDs*" that corresponded to a particular length (in the case of RSA) and, in the case of EC-based keys, whether the points were encoded as strings of a particular length as well.

These checks are not sufficient for the mitigation of a host of invalid public keys, which could result in security degradations. For example, elliptic curve Diffie-Hellman is notoriously susceptible to small subgroup and invalid curve attacks[1], which require mitigation through public key validation. Similarly, RSA public keys could be substituted with invalid keys that do not result as the product of two primes, leading to degradations to the scheme’s effective security.

Furthermore, the lack of checks for the validation of *key ownership* by clients is also responsible for ceding potential for the occurrence of unknown key share attacks, as discussed in 1PW-06-009.

Affected File:
server/src/constraints/validator.go

Affected Code:

```
// InspectPubKey returns true if the Pub Key is valid
func (v *Validator) InspectPubKey(objectName string, attrName string, value
map[string]interface{}) bool {
	switch kid := value["kid"].(type) {
	[...]
	}
	// We could do more here
	return true
}

// InspectECPubKey returns true if the EC Pub Key is valid
func (v *Validator) InspectECPubKey(objectName string, attrName string, value
map[string]interface{}) bool {
	kid, isString := value["kid"].(string)
	if !isString {
		[...]
	}
	if len(kid) != UUIDLength {
		[...]
	}
	x, isString := value["x"].(string)
	if !isString {
		[...]
	}
	if len(x) == 0 {
		[...]
	}
	y, isString := value["y"].(string)
	if !isString {
		[...]
	}
	if len(y) == 0 {
```

[1] https://eprint.iacr.org/2019/526

```
                [...]
        }
        // We could do more here
        return true
}
```

It is recommended to supplement both RSA and elliptic-curve public key validation algorithms with supplementary checks, preferably performed on both client and server. These need to account for the particularities of those public key cryptography schemes.

1PW-06-009 Crypto: UKS attack potential on public key operations *(Info)*

***Note**: After reviewing this issue 1Password does not believe this issue can be presently used to attack customers. 1Password looking to implement mitigations against key sharing in a future version of our API design and implementation as part of its hygiene.*

Whenever a 1Password client uploads an RSA or EC public key to the 1Password server, they are capable of claiming ownership of it. They can also associate them with their account without proving ownership of the key to the server, for example, a challenge-response protocol. Since client public keys are used in sensitive contexts such as vault sharing, the lack of key ownership verification could let a malicious user claim the ownership of another user's public key. This can be done by uploading it as their own, thereby forcing the sharing of vaults with unintended parties.

While the scope of exploitation of this vulnerability appears to be restricted in real-world scenarios, it is nevertheless recommended to impose a quick, nonce-based challenge-response protocol where the server chooses a nonce and requires either a signing or authenticated-encryption-based operation on the nonce. This needs to be performed prior to a key being accepted by the server belonging to a particular client.

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

1PW-06-001 Web: Blocklist bypassable via *X-Forwarded-For* header *(Low)*

It was discovered that the B5 web application's *blocklist* mechanism relies on the content received from the *X-Forwarded-For* header. For each request to the server, the URL path is filtered and checked if it contains strings that are stored in a *blocklist* and blocks the client in case of a match. In particular, this can lead to a bypass of the implemented mechanism if the app runs in an environment without a properly configured reverse proxy and thus the app takes the last IP from the spoofed *X-Forwarded-For* header (see below).

Affected Request:

```
GET / HTTP/1.1
Host: my.b5local.com:3000
X-Forwarded-For: 127.1.2.3
[...]
```

Response:

```
HTTP/1.1 200 OK
[...]
```

During the assessment, it only was reproducible on the local web application, hence the *Miscellaneous* classification. Under certain circumstances, this can lead to a bypass of the blocklist function and it is, therefore, recommended to run the application only behind a well-configured reverse-proxy.

1PW-06-002 Web: Faulty CORS configuration *(Info)*

***Note**: After reviewing this issue 1Password does not believe this issue can be presently used to attack customers. 1Password is looking to improve CORS hardening in the future.*

It was found that some endpoints on *accounts.b5local.com* are running with an overly lax and unrestricted configuration of a Cross-Origin Resource Sharing (CORS) rule. The rules receive the content from the origin's request header and add it to the response header called *Access-Control-Allow-Origin* without any checks (see below). As a result, an external site running on a subdomain is able to send requests via JavaScript to the API. The problem lies in the faulty configuration.

A malicious user might leverage this for information leaks or to alter data, in case the authentication mechanism from the API is based on session-data that is sent via cookies. This might be possible because the *Access-Control-Allow-Credentials* header is set to *true* and is also returned from the API. Based on these settings, browsers will include authentication data - like related cookies - into the request aimed at the API. However, for a successfully authorized request, a subdomain from *1password.com* must be taken over to exploit the improper configuration. Please note that this issue is also reproducible in production environments.

Example Request:

```
GET /api/v1/accountcookies HTTP/1.1
Host: accounts.b5local.com:3000
Origin: https://attacker-controlled.b5local.com:3000
Cookie: b5a-l-
OQXNGQOND5H3NDHDY5JPTICPGY=eyJ0IjoiSSIsIm4iOiJzZWJhYSIsImQiOiJteS5iNWxvY2FsLmNvb
TozMDAwIiwiYSI6IiIsInMiOiIifQ; [...]
```

Response:

```
HTTP/1.1 200 OK
Access-Control-Allow-Credentials: true
Access-Control-Allow-Origin: https://attacker-controlled.b5local.com:3000
[...]

[{"userUuid":"5SLSYUHKG5HTJANPYRU2DOMV3Y","userEmail":"seba+1pw@cure53.de","user
Name":"sebaa","userAvatarURL":"","usingMyDomain":true,"accountUuid":"OQXNGQOND5H
3NDHDY5JPTICPGY","accountType":"I","accountName":"sebaa","accountDomain":"my.b5l
ocal.com:3000","accountAvatarURL":"","hasPrioritySupport":false}]
```

While this poses no danger at present, it is nevertheless recommended to rethink the need for deploying the CORS headers with this configuration. One possible measure is to restrict access via an allow-list-based filter. Therefore, the relevant CORS header should only be returned if the origin of a request matches an allow-listed domain.

1PW-06-005 Crypto: Private key remains static after vault password reset (*Medium*)

Note*: 1Password is aware of this issue and has previously documented it. 1Password considers this an issue that is important for protecting customers in the future, but considers the present day impact of this issue to be limited. Mitigating and resolving this issue is being worked on as part of future 1Password security designs.*

This issue is re-filed from the 1PW-01 audit due to its continued pertinence to 1Password. It is being cited in other parts of this report because no remediation took *place since the original filing date. Due to the issue's lack of novelty, it is here included as a miscellaneous item.*

It was found that changing the master password for the 1Password vault does not reset its underlying RSA key-pair. The latter provides the set of secret keys used to encrypt individual vault items. Therefore, this flaw means that in the event of a vault RSA key compromise (to which multiple vectors are offered through *1PW-01-015*), it is impossible to restore the vault to a "*safe*" state.

It is recommended to ensure that changes to user-passwords made for the vault permeate across the vault's key space. In other words, all keys that could be leaked during a compromise should be rotated out of usage in future editions of the vault in question.

1PW-06-006 Crypto: 1Password vaults vulnerable to server compromise *(Medium)*

***Note**: 1Password is aware of this issue and has previously documented it. 1Password considers this an issue that is important for protecting customers in the future, but considers the present day impact of this issue to be limited. Mitigating and resolving this issue is being worked on as part of future 1Password security designs.*

This issue is re-filed from the 1PW-01 audit due to its continued pertinence to 1Password today. It is being cited in other parts of this report since no remediation took place since the original filing date. Due to the issue's lack of novelty, it is here included as a miscellaneous issue.

It was found that almost all of the protocols outside of the main 1Password vault encryption and synchronization protocol use non-authenticated Diffie-Hellman key exchanges. This includes:

- **Vault Sharing**, which allows 1Password users to share access to vaults with team members or other 1Password users.
- **Recovery Contacts**, which allows 1Password users to specify other users that, in case of an emergency, can obtain access to the 1Password vault.
- **User Transfer**, which allows 1Password users to transfer account data to another account.

As such, it is possible for a malicious or compromised 1Password server to obtain decryption keys for 1Password vaults, allowing decryption not only of past vault items, but also of future vault items. The latter is possible due to *1PW-01-014*.

1PW-06-010 Crypto: Malfunctioning and unused cryptography components *(Info)*

It was observed that the 1Password Go codebase contained a "*dhke*" submodule that, while unused, might offer risky and malfunctioning cryptographic components, specifically in the event that it is used again in the future. Some examples of the relevant scenarios can be found below.

- The module implements a broken signature API in *signature.go* which does not perform signature checks.
- The module implements *"key verification"* functionality in *keyverify.go* that appears to intend the mitigation of Man-in-the-Middle attacks on keys but which would not accomplish any improvement in that respect.
- The module appears to re-implement secure CSPRNG functionality in a way that ends in a conflict and is redundant with the *crypto/rand* module offered by the Go standard library.

Affected Path:
b5/vendor/go.1password.io/xplatform-security/dhke

It is recommended to simply delete the *dhke* module from the codebase. The only two functions used from the *dhke* module are:

- *dhke.Base64URLToBig*
- *dhke.BigToBase64URL*

These functions can be migrated into the namespace of a utility *functions* submodule, as they have nothing to do with Diffie-Hellman key exchange. The rest of the *dhke* module may then be safely deleted from the codebase.

Conclusions

As noted in the *Introduction,* Cure53 needs to emphasize a lot of strength and positive indicators observed on the 1Password B5 web application and permission enforcement components. While ten findings, including the *High* severity flaw, cannot be disregarded, four members of the Cure53 team confirm that the examined application made a strong impression with regard to security. Across the respective WPs completed over the course of fourteen days in October 2020, the 1Password B5 complex seems to have successfully reached the key security milestones.

Thorough and full coverage resulted in five vulnerabilities and five general weaknesses confirmed and documented. The issue which received *High* impact markers illustrates that not all areas of the application are stable against more complex attack scenarios. However, its existence does not indicate that the application is in a bad or unstable state. On the contrary, the 1Passwod B5 application makes a very capable impression and withstands common attack-types. This is also reflected in the result of the audit, which mainly consists of findings that could be uncovered through deeper analyses only.

Commenting on the detailed procedures, the main priority was assigned to tests focusing on the identification of the web application- and server-side problems affecting the B5 application. An explicit focus was placed on possible ACL implementation flaws within the tested compound, while possible leaking of potentially sensitive information and parsing issues were also addressed in great depth. The basic idea behind this Cure53 investigation was to find out whether the existing functionality of the endpoints and its environment can be deemed healthy enough to withstand attacks by malicious users.

With the focus on typical modern application problems, the issues connected with various types of injection attacks, which could compromise the server part of the application, were investigated without significant success. Also, Cure53 explored the relevant API and functionality. A special focus was also given to the authentication and authorization of vault accesses. In this realm, adding the access type to almost all function prototypes makes malicious cross-account accesses very unlikely. In contrast, it was found that 1Password is more prone to inner-account accesses that translated to the finding in 1PW-06-007.

On the one hand, Cure53 is happy to report that the general audibility of 1Password is - due to the transparent code and well-written source code - very good. On the other hand, the overwhelming complexity of all functionality accommodates a lot of exploit potential. This fact is underlined by the presence of multiple actions that partially perform the same work with different restrictions, as seen in 1PW-06-007. It is advisable to further increase the code reuse and keep up the strict separation between safe and unsafe actions. It could be helpful to include the restrictions of an action within that action whenever possible.

Although it is possible to break single primitives of the 1Password's security concept, the overall security level remains strong because:

- Passwords in a safely established vault are very hard to compromise for external attackers.
- Vulnerabilities in the access control cannot leak the vault's encryption key.
- An exposed vaults encryption key requires proper access control to expose the individual items.

At the same time, the test revealed weaknesses in API endpoints that return user-data. The affected endpoints expose sensitive information from users, such as email addresses, UUIDs, or info on whether 2fA is enabled (see 1PW-06-004). Additionally, information about groups and vault memberships of users is revealed, which is very useful for further attacks, such as 1PW-06-007. It is paramount that the data should only be available for administrators.

The newly implemented brute-force protection in the *verification* step from the user-registration is bound to the sent email address. This can be used by attackers to prevent users from finishing the registration process and, thus, making it impossible for them to use the 1Password service more broadly (see 1PW-06-003). This risk should also be resolved as a priority, so that new users can no longer be excluded from the platform.

Cure53 needs to underscore the absence of a number of issues connected with injection attacks, which could compromise the client-side part of the platform. This is a quite rarely found positive indicator. The client-side benefits from the properly used React framework and DOMPurify, so no issues were found in these realms. As a result, the B5 web application made a robust impression with regard to client-side vulnerabilities.

A comprehensive review was conducted of the cryptographic implementations and operations performed on both the client and server side of B5. Due to the frequency of 1Password cryptography audits, the expectation was to find only minor issues (if any) and this expectation was generally met. Unfortunately, 1Password continues to neglect issues reported in its public key cryptography-based functionalities in the past, for instance as regards vault sharing. This leads to the re-reporting of 1PW-06-006 and 1PW-06-005.

New albeit related issues linked to public key cryptography in 1Password were also discovered. Both 1PW-06-008 and 1PW-06-009 stem from insufficient client- and server-side public key validation, which happens before those public keys are allowed to be used for sensitive features such as vault sharing. Finally, unused but potentially dangerous cryptographic code was found within the 1Password shared codebase. It is recommended to simply remove this code, as documented in 1PW-06-010.

All new issues were reported, discussed and confirmed with the 1Password team during the completion of this report. While the 1Password cryptography stack may be considered mature, the 1Password's approach to public key cryptography remains suboptimal in the long-discussed respects. It is recommended to invest more effort in addressing these issues.

To conclude, the application has accomplished security at a high level. This has two main causes. First, very good security architecture and well-implemented defense mechanisms manage to fend off attacks and prevent problems by default, Second, the strength comes from the continuous execution of audits. This is an especially good two-pronged approach because this password manager has a high degree of complexity and offers many functionalities, which can also lead to more complex problems that are not visible at first glance. This conclusion applies to the *High* severity finding, which should be treated as a matter of urgency to make sure that guest users can be protected from data theft in future releases.

The structure of the source code is also particularly praiseworthy. This not only leads to good readability, but also demonstrates the experience and expertise of the software architects, developers and security engineers involved. All in all, very few exploitable issues were spotted beyond the resurfacing cryptography-related findings. Cure53 sees the 1Password B5 application complex as being still on the right track to deliver a secure foundation in their customer services. At the same time, this project draws attention to certain aspects where work still needs to be done in order to move the platform forward to an even better security premise.

Cure53 would like to thank Mahdi Yusuf, David Girvin, Adam Caudill, Rick Fillon, Rob Yoder and Cyrus Lee from the 1Password team for their excellent project coordination, support and assistance, both before and during this assignment.