Fine penetration tests for fine websites

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

Pentest-Report RememBear 08.2017

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. Abraham Aranguren, Dipl.-Ing. Alex Inführ, BSc. Christoper Kean, Norman Hippert, Nadim Kobeissi

Index

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

Introduction

This report documents the findings of a penetration test and source code audit against the RememBear password manager. The project was carried out by Cure53 in mid-August 2017 and yielded a total of fifteen security-relevant results.

It should be noted that this security assignment had a very wide scope. The project was granted a time budget of twenty-five days, which was then split into six separate modules described below in the "*Scope"* section. Besides auditing sources and performing a classic penetration test, the Cure53 team engaged in a thorough cryptographic review of the RememBear software. The latter was aimed at verifying whether all cryptographic components can be assessed as well-written and working as intended.

Based on the required expertise and skillsets, six members of the Cure53 team were involved in the completion of this project. As for the approach, a white-box methodology was chosen and the Cure53 testers received access to all relevant source code materials as well as binary builds. Additional information and feedback was exchanged quickly between the testers and the RememBear in-house team to ensure a good test coverage. This was assisted by having a dedicated Slack channel, which ultimately helped overcome arising technical issues in a prompt manner. A good communication setup turned out indispensable, as the initial phase of the project needed to address broken builds. After this slightly bumpy start, the tests proceeded smoothly and efficiently.

As already indicated, the Cure53 testers managed to spot fifteen security risks. The issues were further categorized into five vulnerabilities and ten general weaknesses. It is important to emphasize that not a single finding warranted a "Critical" ranking in terms of severity or impact, while only two problems were deemed to have a "High" level of security implications. The report will now shed light on the detailed aspects of the scope, particularly in terms of how six sub-modules were developed. It then proceeds to a case-by-case discussion of each finding, offering technical descriptions and mitigation advice when applicable. The closing paragraphs are devoted to concluding remarks and impressions about the general security of the RememBear product.

Note: *For this public version of the RememBear pentest report, several code snippets, screenshots and attack examples have been removed, as requested by the TunnelBear team.*

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

Scope

- **Module 1: OSX Client**
  - Code Audits and Penetration Tests against OSX Mobile App & available Sources. Relevant code and binaries were made available to Cure53.
- **Module 2: iOS App**
  - Code Audits and Penetration Tests against iOS Client & available Sources. . Relevant code and binaries were made available to Cure53.
- **Module 4: Android App**
  - Code Audits and Penetration Tests against Android Mobile App & available Sources. Relevant code and binaries were made available to Cure53
- **Module 5: Browser Extension**
  - Code Audits and Penetration Tests against Browser Extension & available Sources. Both code and binaries were made available to Cure53.
- **Module 3: Windows Client**
  - Code Audits and Penetration Tests against Windows Client & available sources.
- **Module 6: Backend, API & Crypto**
  - Code Audits of the available Scala Backend Code & Rust Sources, API Tests. Relevant code and binaries were made available to Cure53.

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

RMB-01-001 Mac/iOS/Android/Win: Faulty domain detection leaks passes *(High)*

As any password manager, RememBear is evaluates each loaded URL and matches it against stored vault items. If a user has saved a username and password combination for the URL, the application offers a possibility to automatically fill and submit the credentials. However, it was discovered that the currently deployed algorithm for URL parsing is faulty. Specifically, the algorithm attempts to detect and remove top level domains to extract the *host* part. This is because the current design treats subdomains in the same way as the main domain.

It can be observed that since the algorithm is removing up to two top level domains, it actually treats *victim**.co.uk***, *victim**.com***, *victim**.de*** and even ***test.**victim**.co.at*** as if they were identical. By this logic, it proposes to autofill the same user-credentials for all these

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

domains as if they were equivalent. This behavior could trick a user into revealing their credentials, as they may end up submitting them to an attacker-controlled website.

The described behavior can be found on the Windows and Mac platforms in the Chrome extension, as well as in the vault software. Moreover, it is also present in both the mobile RememBear browser application and in the native browser.
In order to solve this problem, the entire hostname should ideally be used for the process of credential storage matching. This is the simplest solution and would completely eliminate attacks against the core logic employed in the hostname matching algorithm at present. If this approach is not feasible, it is recommended to review the domain validation process and ensure that the last two or three components of the hostname are used, making the process dependent on the received value.

To clarify, some examples on handling the matter are specified next.

- If a domain like *victim.co.uk* or *www.victim.co.uk* is received, the entire three last sections of the hostname should be used, i.e. *victim.co.uk*.
- If a domain like *victim.com* or *www.victim.com* is received, then the last two sections should be sufficient, i.e. *victim.com*.

Finally, if neither of these approaches can be implemented, another option would be to review the domain validation process and ensure that ccSLDs are detected properly. Although there is no complete list of ccSLDs available, a good starting point can be the public suffix list maintained by Mozilla[1]. As this list is mostly maintained for browser purposes - like cookies - it requires some manual work when one seeks to extract ccSLDs, which are conversely important in a specific context of a password manager rather than generally for browsers.

RMB-01-006 Android: Denial of Service issue in Login Model *(Low)*

The Android RememBear app is exporting three activities which can be invoked by any other application. The *LoginActivity* activity accepts a parceled class via the *send intent*, and it retrieves *Master Password* stored in this class. This data is then used as a default value in the GUI. Serialization/parceling can be abused to cause a crash in the receiving app, with the process happening upon sending an unavailable or a malformed class. Additionally it must be noted that the RememBear app is supporting Android versions 4.3.1 or newer. A vulnerability in the parcel/serialization code on Android was discovered to affect all versions up to 5.1.1[2]. Consequently, this can be abused to cause RCE inside the application and threatens the overall security of the RememBear app.

[1] https://publicsuffix.org/list/public_suffix_list.dat
[2] https://www.usenix.org/system/files/conference/woot15/woot15-paper-peles.pdf

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

It is recommended to modify the *LoginActivity* class and remove the code responsible for unparceling. Given the implemented functionality of the existing activity, it is possible to obtain the necessary data by solely relying on simple strings sent via an intent. This ensures that older Android versions cannot be attacked via the previously mentioned parceling/serialization vulnerability. Furthermore, it would actually be possible to completely remove the activity in question, or at least stop exporting it to third-party applications, especially since it does not appear to be implementing important user-features. Consideration should also be given to removing the BROWSABLE category of the *intent-filter,* so that the activity is not reachable for websites.

RMB-01-007 Mac: RememBear app DoS via URL handler *(Medium)*

It was found that the *RememBear* Mac application exposes a URL handler to third-party apps. When the URL handler is invoked incorrectly, this will consistently crash the Mac OS X app. A malicious app could leverage this weakness to repeatedly and constantly crash the RememBear app while running in the background. Malicious websites can also accomplish this but the impact would be more limited due to the user being prompted first.

It is recommended to review the business need of exposing this URL handler. If possible exporting it to third-party apps should be ceased. Another possibility would be to implement adequate exception-handling. If the URL handler must remain exposed to third-party apps, additional mitigation measures should be considered to prevent user-disruption via arbitrary URL handler invocations.

RMB-01-010 Mac/Windows: DoS via RememBear Helper listener *(Low)*

It was found that the *RememBear Helper* service of the Mac and Windows apps currently listens to on all network interfaces. A malicious attacker on the local network could leverage this weakness to send malformed traffic to this interface in order to make the entire Mac OS X machine perform very poorly with up to 100% CPU spikes. On Windows the DoS potential does not appear to be as great, as only about 10% CPU is used. However, in both cases an attacker on the local network can attempt to connect and interact with the *RememBear Helper* service. In this realm, it can be imagined that an attacker gets paired up with the vault if the user miss-clicks on the "*Yes*" button of the pairing process. An attacker on the local network could also try to initiate the pairing process repeatedly until the user accepts additional chances of accessing the password vault.

The main reason behind this issue is tied to the way in which the *RememBear Helper* is listening. More specifically, because the service is listening on all network interfaces, an attacker in the local network can establish connections to the service.

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

It is recommended to alter the RememBear listeners so that they only listen on the *localhost* interface. An abuse mechanism should ideally be implemented so that malformed requests are banned with as little performance impact as possible. For example, a malicious IP or origin (if a similar attack is attempted from a website) could be banned before any processing takes place.

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.

RMB-01-002 Mac: Insecure ATS configuration could lead to leaks/MitM *(Info)*

It was found that the Mac app currently deploys an insecure ATS configuration. This issue is merely "Informational" as far as risks are concerned because the RememBear app implements Pinning. As a result, MitM attacks against HTTPS connections would not be possible. However, if Pinning protections were to be removed in the future, this configuration would allow the app to establish clear-text HTTP connections to the backend server. Furthermore, it would signify accepting invalid TLS certificates for HTTPS connections. A malicious attacker with the ability to MitM network communications could leverage this weakness to observe and modify clear-text HTTP and HTTPS traffic.

The official Apple documentation has the following to say about *NSExceptionAllowsInsecureHTTPLoads*[3]:

> *"With this key's value set to YES, your app can make secure connections to a secure server but* ***can also connect insecurely to a server with no certificate, or a self-signed, expired, or host-name-mismatched certificate.****"*

It is recommended to remove all ATS exceptions and use TLS during development. This would remove the need of having exceptions during development, which would henceforth mean that having a process to remove the insecure settings before production deployment would no longer be necessary. If this approach is not viable, it is important to introduce a process that ensures that these insecure settings cannot be applied to the production version as a result of human error. Ideally, this should be done in an automated fashion to reduce the likelihood of a developer forgetting to disable insecure settings by mistake.

[3] https://developer.apple.com/library/content/documentation/Ge...eference/Articles/CocoaKeys.html

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

RMB-01-003 API: Timing Side-Channel on SRP Authentication *(Low)*

During the code audit it was discovered that the SRP *m1* comparison is not safe against timing attacks[4]. This could lead to advanced brute-force techniques, notably reliant on timing differences[5] to guess the necessary value. However, due to more information needed to even arrive at this early stage, the attack scenario is perceived as not realistically exploitable.

Consideration should be given to using a constant time comparison function for handling the comparison of the SRP *m1* value.

RMB-01-004 iOS: Copied login passwords are readable by other apps *(Low)*

It was discovered that the text fields of the *Master Password* and the *Password* fields in the *Login Item* creation menu disallow copying sensitive data. However, the *password* field of existing login items enables copying passwords to the general pasteboard, from where they are readable to other applications: Before iOS 9, applications running in the background could monitor *UIPasteboard.general.string*, as this system's pasteboard contents were only accessible to apps in the foreground[6].

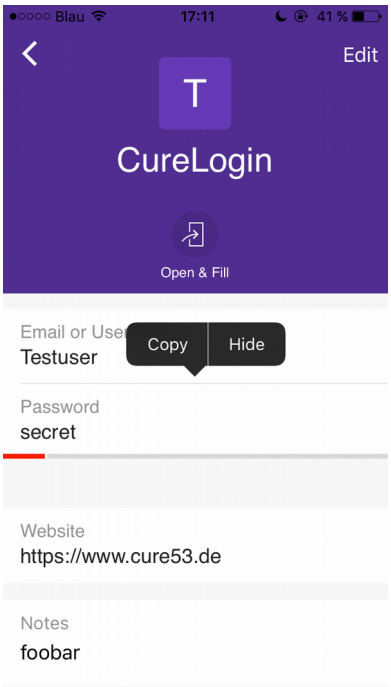

Fig.: Password field employs the copy operation on the general pasteboard

[4] https://codahale.com/a-lesson-in-timing-attacks/
[5] https://events.ccc.de/congress/2011/Fahrplan/events/4640.en.html
[6] https://github.com/OWASP/owasp-mstg/blob/master/Docu...#testing-for-sensitive-data-in-the-clipboard

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

For this reason, the general pasteboard should ideally never be used for copy/cut-operations performed on sensitive data. It is recommended to use an application-specific custom pasteboard to transfer credentials to the BearOwser app internally. With regard to the usage of RememBear in Safari, disabling the copy function in the extension could be considered. Instead, relying on the existing "Open & Fill" functionality to pass on credentials seems like a more security-conscious option. For more information, please review the *UIPasteboard* article provided by Apple[7].

RMB-01-005 Mac: *RememBear* and *RememBearHelper* are not sandboxed *(Info)*

It was found that the Mac app fails to leverage the Mac OS X App Sandbox at present. This means that vulnerabilities in the *RememBear* app and the *RememBearHelper* can provide access to the full permissions of the user in the file-system, memory, etc. In essence, this makes the RememBear Mac app a more attractive target for privilege escalation.

Please note that the Apple documentation contains some high level comments on this behavior[8]:

> *"Apps distributed through the Mac App Store must adopt App Sandbox. Apps signed and distributed outside of the Mac App Store with Developer ID can (and in most cases should) use App Sandbox as well."*

It is recommended to use the Mac OS X Sandbox and this can easily be accomplished with Xcode. A careful review of the Apple *App Sandbox Checklist*[9] will ensure a smooth transition within this process.

RMB-01-008 Windows: *Password save* dialog uses wrong domain *(Low)*

The Windows implementation differs from the mobile applications regarding the detection of framed and known vault items. When a website frames another web page with a stored login resource, it offers the user a possibility to autofill the stored credentials. It was discovered that when a user employs the autofill feature in a framed web page, the Windows application shows a "*Do you want to remember this login?*" dialog. However, it does so in the context of the top level domain. In case a user opts for clicking on *"Remember",* the credentials of the framed page get stored for the top level domain, too. Interestingly the Mac extension is not affected by this issue as it opens a new window when the form is submitted. This results in the Extension not showing the *save password* dialog.

[7] https://developer.apple.com/documentation/uikit/uipasteboard
[8] https://developer.apple.com/library/content/documentatio.../AboutAppSandbox/AboutAppSandbox.html
[9] https://developer.apple.com/library/content/documentation/Security/...8-SW1

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

It is recommended to implement a dedicated check in the Windows application and this way detect the submission of the stored credentials inside an Iframe. This will help prevent the creation of the "*Store password"* dialog for the wrong domains.

RMB-01-009 Windows: *DisposableString* provides no added Protection *(Info)*

During the code review it was discovered that the Windows application heavily uses the *DisposableString* class, which is derived from the *SecureString*[10] implementation in Mono. As this method rarely employs the necessary *Marshall.AllocHGlobal* to protect data against ending up in the "garbage collector", the benefit of this class is minute. However, an attacker with access to the server can still monitor the system's memory and might extract passwords. A better implementation can be found in the *.NET Core Library*[11] and should be used instead, as it guarantees both keeping the data in the unmanaged memory, and zeroing it out properly.

As this application runs on Windows, it should be considered to use the proper *SecureString* class[12] which provides appropriate protection and adequate memory encryption. Modifying critical security classes should be done with as much care as possible, with an ideal solution being to avoid it altogether.

RMB-01-011 Rust: Passphrase Generation and Validation Recommendations *(Info)*

The Rust cryptography core in RememBear uses a complex set of functions in order to calculate and validate users' passwords and passphrases. The actual contents of this implementation are not ideal because they fail to present added security benefit despite heightened complexity.

It is recommended to simplify the generation procedure. Specifically, it should be considered to simply create an alphanumeric, all-case character-set *c* which totals 62 characters. Measuring entropy out of this character set is much simpler and more reliable than the current implementation. In order to determine how many characters need to be sampled, for example for a password with 80 bits of entropy, simply calculate $log(2^{80}) / log(62) =\sim 14$. For 128 bits of entropy: $log(2^{128}) / log(62) =\sim 22$.

A validation procedure then becomes unnecessary.

RMB-01-012 Public Key Protocol Recommendations for Browser Extensions *(Info)*

The RememBear protocol overview document (in Version 3) illustrates a public-key/authenticated key establishment protocol that operates between three principal agents: the RememBear browser extension, a background application, and the Rust

[10] https://github.com/mono/mono/blob/master/mcs/class/corlib/System.Security/SecureString.cs
[11] https://github.com/dotnet/coreclr/blob/master/src/mscorlib/shared/System/Security/SecureString.cs
[12] https://msdn.microsoft.com/en-us/library/system.security.securestring(v=vs.110).aspx

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

cryptography core. All three principals are located locally on the user's computer.

The goal of this protocol is to encrypt and authenticate all communications between the three principals so that hijacking of the WebSockets' layer exposed by the background application cannot reveal any data being communicated to the browser extension.

The protocol is only adequate if no third-parties can passively monitor the WebSocket connection between the browser extension and the background process. In the case of passive monitoring being attainable, an attacker can successfully compromise the channel. This is because the authentication string used as part of the protocol has only 32 bits of entropy. Here is how a potential attack could work:

- Evil browser extension `E` listens on the WebSocket `W`, waiting for a connection by using `*onBeforeRequest*`.
- Benign browser extension `B` wants to connect to background process `P` using `W`.
- `E` detects that `B` tried to use `W`. It ends that connection and instead starts its own link with `P` over `W`. This forces `P` to generate `*pk2*` and communicate it to `E`.
- `B` tries to reconnect to `W`. Since `*pk2*` is static, it receives the same `*pk2*`.
- If `E` can monitor what `B` is sending over `W` at this stage, it will know `*pk1*`.
- `E` trivially obtains a colliding *auth* string for `*pk1*|*pk2*` using a `*pk1*'` that it controls.
- `E` again intercepts the connection immediately after `B` shows the *auth* string for confirmation to the user.
- At the end,`B` was tricked into making the user confirm the *auth* string for a malicious key. The channel is compromised.

The attack enumerated above is not currently believed viable, hence the classification of this issue stands at "*Info*". It is nevertheless recommended to increase the complexity of the authentication string and have it use twelve rather than just six characters as a precaution.

RMB-01-013 Custom TLS Protocol Recommendations *(Info)*

The RememBear protocol overview document (in Version 3) illustrates a transport layer encryption protocol that essentially amounts to "TLS over TLS". The point of this protocol is to "*reduce the impact of Cloudbleed-like incidents*".

While the implementation does not appear to be defective, it is questionable whether the serious engineering overhead is warranted. "Cloudbleed-like incidents" could be more easily avoided by investing in a more trustworthy low-level TLS stack, and, especially,

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

avoiding the use of CDNs for security-sensitive connections. A recommended OpenSSL stack is *BoringSSL*[13], the security-focused OpenSSL alternative developed by Adam Langley.

RMB-01-014 WebExtension: DOM Clobbering can influence Content Script *(Info)*

The assessment revealed that the WebExtension, especially the content script, does not always consider *DOM Clobbering*[14] upon interacting with attacker-controlled HTML code. *DOM Clobbering* allows to overwrite certain functions of HTML elements, especially form elements, by specifying the desired function's name in the *name* attribute of a child element. Although it was not possible to cause a vulnerability in the WebExtension with this approach during testing, it is important to keep this threat in mind while developing the extension further.

It is recommended to properly check the type of each attribute or function before interacting with attacker-controlled HTML elements inside the content scripts. Although this behavior did not yield any vulnerabilities in this test instance, it increases the risk of being vulnerable in the future. What is more, it must be noted that a similar behavior is possible for global and undefined variables inside the content script. In case the script determines certain code paths by evaluating whether a global variable is defined or undefined, a malicious website can specify these variables via HTML element IDs. This is because the latter are stored in the global window object of the content script, therefore having the power over influencing the code path of the content script.[15]

It is recommended to properly initialize all of the global variables used inside the content script.

[13] https://boringssl.googlesource.com/boringssl/
[14] http://www.thespanner.co.uk/2013/05/16/dom-clobbering/
[15] https://bugs.chromium.org/p/project-zero/issues/detail?id=1225

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

Conclusions

The overall outcome of this Cure53 summer 2017 security assessment point to a notable security dedication at the RememBear project. The tests, which were completed by six members of the Cure53 team over the course of twenty-five days, yielded fifteen security-relevant discoveries.

Before moving on to the technical aspects of the results, it should be noted that some minor test limitations were present for this assignment. First of all, the project was granted a relatively limited time budget. This is particularly vital given the large codebase and extensive functionality. Notably, iOS, Android, Windows and Mac clients, browser extension, source code analysis, as well as crypto and design review were all included as test components in the scope. Secondly, it has been noticeable that the development process of the RememBear suite was affected by tight deadlines. Evidencing this was the fact that builds were generally provided only one or two days before actual testing started, leaving little room for in-depth reconnaissance. Thirdly, there were certain subtle functional bugs. Although these were minor, they managed to impact and disrupt testing on a few occasions. This applies, in particular, to the Android, iOS and Mac clients, which turned out to require long hours of debugging and finding workarounds before being test-ready. Additional troubleshooting activities were also warranted around installing the Windows client.

Having discussed the setup limitations, it should be nevertheless emphasized that the core aspect - that is security of the RememBear products - was found to be at the rather high level. All tested clients provided robust impressions and no major flaws were identified. Importantly, backend network communications were generally protected with Pinning in an appropriate manner, with the sole exception of risks found on the Amazon S3 traffic. More specifically, the lack of Pinning on Amazon S3 opens doors to ZIP bomb and file overwrite attacks against all clients. While succeeding with this approach requires high-profile and powerful attackers with the capacity to intercept network communications with a valid CA certificate trusted by the operating system, it nevertheless calls for urgent attention. Further note that the configuration of the Amazon AWS production configuration was not in scope for this test.

On a general level, it should be underlined that the WebExtension is built well, with neither unnecessarily exported extension resources, nor major vulnerability detected. Conversely, the broken domain detection is a design problem on all components. In essence, the backend utilized modern technologies and frameworks, thus providing a good baseline security. Let us know shed light on three aspects of key vulnerabilities, minor weaknesses, and cryptographic issues.

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

As far as the actual five security vulnerabilities discovered during testing are concerned, one important point to make is that not a single problem was deemed to be of a “Critical” severity or security implications. For the two issues ranked as “High”, the first problem had to do with a design flaw around the autofill functionality and incorrect handling of top level domains (see RMB-01-001). All remaining flaws revolved around different Denial of Service attacks, ranging from possible RCE against the Android app via deserialization of user-input (RMB-01-006), to DoS against the Mac app via URL handler (RMB-01-007), to increased resource consumption and possible vault interaction on two clients (RMB-01-010).

When looking at miscellaneous issues, one should note the prevalence of flaws marked with “Low” or even “Informational” severity rank. In the more important first class of issues, the observed slight shortcomings were tied to timing side-channel weaknesses on the SRP authentication of the API (RMB-01-003), possible password leakage via the iOS clipboard (RMB-01-004), as well as *save password* dialog weaknesses (RMB-01-008). Findings documented as “Info” for the sake of completeness provide advice on, among others, improving ATS configuration and sandboxing on Mac, considerations around the *DisposableString* on Windows, and avoiding DOM Clobbering on web extension. The Cure53 team furnishes additional note on fine-graining passphrase generations, deployment of public keys and TLS, all viewed as defense in-depth rather than indispensable mechanisms.

To finalize with some general remarks on the examined cryptographic implementations, the symmetric primitives in the Rust core and the clients were sound and well-implemented. They are grounded in the deployment of *libsodium*, which is a state-of-the-art cryptographic library. While various notes and suggestions were collected during the cryptographic analysis, no severe *implementation*-related vulnerability was spotted. In other words, it is believed that a real-world attacker would remain powerless in face of the employed defense mechanisms. The PKI design for securing communications between the browser extension and the local backend could be modernized by relying on the Curve25519 key generation and authentication on both sides. This accomplishes the matching and ambitious security goals while it eradicates the added complexity of RSA and X.509 certificate parsing. The protocol design documents need to be expanded with stronger detail and formalism. In some instances, the lack of rigor in specification leads to strong openings for attacks. Finally, while the SRP protocol implementation guarantees some resistance against offline attacks, additional research could determine whether these properties could also be acquired through a simpler protocol.

All in all, the RememBear is a robust and promising project. Though some minor security issues shall still be resolved to benefit the already strong design, the Cure53 testing team faced a well-hardened entity, clearly developed from a security-conscious stance.

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

Cure53 would like to thank Thomas Cordua-von Specht, Dane Carr, Rodrigue Hajjar and the TunnelBear / RememBear team for their excellent project coordination, support and assistance, both before and during this assignment.